Basic

Evaluation Criteria

- Scribe - 5%
- Weekly Quizzes - 20%
- Mid term 1 - 15%
- Mid term 2 - 15%
- Final Exam - 45%

Protocol

A protocol defines the format and the order of messages exchanged between two or more communicating entities, as well as the actions taken on the transmission and/or receipt of a message or other event.

Packet Switching

- End systems exchange messages with each other.
- Long messages into smaller chunks of data known as packets.
- Between source and destination, each packet travels through communication links and packet switches
- Packets are transmitted over each communication link at a rate equal to the full transmission rate of the link. So, if a source end system or a packet switch is sending a packet of L bits over a link with transmission rate R bits/sec, then the time to transmit the packet is L/R seconds.
- Each packet can transfer via different path.

Store And Forward

Store-and-forward transmission means that the packet switch (router) must receive the entire packet before it can begin to transmit the first bit of the packet onto the outbound link (Maybe because we need to process header). Router can store multiple packets.

Question: Consider a simple network consisting of two end systems connected by a single router, as shown in the figure.

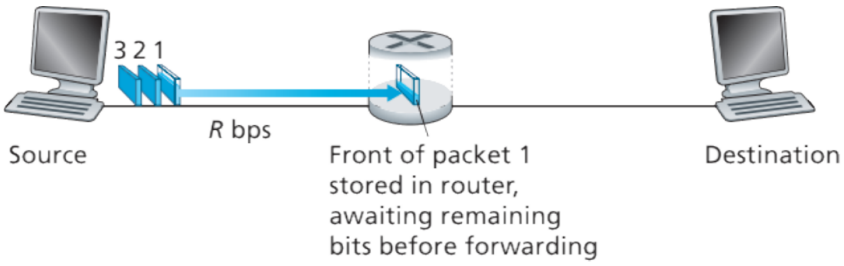


Figure: [Kurose and Ross] Store-and-forward packet switching.

All links in the above figure have transmission rate R bits/sec. The source wants to send 4 packets of L bits each to the destination. If the source starts transmission of the first packet at time 0, at what time would the destination receive all the packets?

Answer: $2L/R + 3L/R = 5L/R$.

Delay in Packet-Switched Networks

- **Processing Delay:** The time required to examine the packets header and determine where to direct the packet is part of the processing delay. The processing delay can also include other factors, such as the time needed to check for bit-level errors in the packet that occurred in transmitting the packets bits from the upstream node to router A. Processing delay in high-speed routers are typically on the order of microseconds or less. After this nodal processing, the router directs the packet to the queue that precedes the link to router B.
- **Queuing Delay:** At the queue, the packet experiences a queuing delay as it waits to be transmitted onto the link. The length of the queuing delay of a specific packet will depend on the number of earlier-arriving packets that are queued and waiting for transmission onto the link. If the queue is empty and no other packet is currently being transmitted, then our packets queuing delay will be zero. On the other hand, if the traffic is heavy and many other packets are also waiting to be transmitted, the queuing delay will be long.Queuing delays can be on the order of microseconds to milliseconds in practice.
- **Transmission Delay:** Assuming that packets are transmitted in a first-come-first-served manner, as is common in packet-switched networks, our packet can be transmitted only after all the packets that have arrived before it have been transmitted. Denote the length of the packet by L bits, and denote the transmission rate of the link from router A to router B by R bits/sec. The transmission delay is L/R. This is the amount of

time required to push (that is, transmit) all of the packets bits into the link.

- **Propagation Delay:** Once a bit is pushed into the link, it needs to propagate to router B. The time required to propagate from the beginning of the link to router B is the propagation delay. The bit propagates at the propagation speed of the link. The propagation speed depends on the physical medium of the link. The propagation delay is the distance between two routers divided by the propagation speed. That is, the propagation delay is d/s, where d is the distance between router A and router B and s is the propagation speed of the link. Once the last bit of the packet propagates to node B, it and all the preceding bits of the packet are stored in router B. The whole process then continues with router B now performing the forwarding.

Circuit Switching

- Resources needed along a path (buffers, link transmission rate) to provide for communication between the end systems are reserved for the duration of the communication session between the end systems.
- A circuit in a link is implemented with either frequency-division multiplexing (FDM) or time-division multiplexing (TDM).
- In **FDM**, the frequency spectrum of a link is divided up among the connections established across the link. Specifically, the link dedicates a frequency band to each connection for the duration of the connection.
- **TDM** is considered to be a digital procedure which can be employed when the transmission medium data rate quantity is higher than the data rate requisite of the transmitting and receiving devices. In TDM, corresponding frames carry data to be transmitted from the different sources. Each frame consists of a set of time slots, and portions of each source is assigned a time slot per frame.
- Circuit Switching is basically used for real time applications
- Packet switching is better than circuit switching as even traditional applications of circuit switching such as phone calls is now being done with the help of packet switching which even allows for both of the things, i.e. internet and phone call to happen simultaneously (VoLTE). Also it can easily accommodate more sporadic users without change in configuration.

Protocol Layering

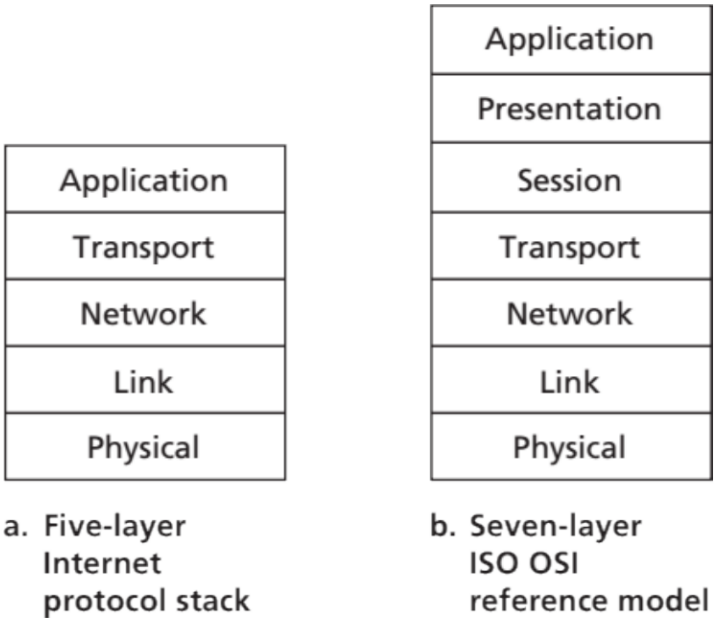


Figure: [Kurose and Ross] The Internet protocol stack (a) and OSI (Open Systems Interconnection) reference model (b).

Layer interacts with layers only next to it.

Benefits of layering: * We can modify a layer freely, i.e. tweak it. * Layer acts as a wrapper which can be unwrapped at the other end.

Application Layer

The application layer is where network applications and their application-layer protocols reside. (Like HTTP, FTP, DNS, etc) An application-layer protocol is distributed over multiple end systems, with the application in one end system using the protocol to exchange packets of information with the application in another end system. We'll refer to this packet of information at the application layer as a message.

Transport Layer

The Internet's transport layer transports application-layer messages between application endpoints. In the Internet there are two transport protocols, TCP and UDP, either of which can transport application-layer messages. TCP provides a

connection-oriented service to its applications. This service includes guaranteed delivery of application-layer messages to the destination and flow control (that is, sender/receiver speed matching). TCP also breaks long messages into shorter segments and provides a congestion-control mechanism, so that a source throttles its transmission rate when the network is congested. The UDP protocol provides a connectionless service to its applications. This is a no-frills service that provides no reliability, no flow control, and no congestion control. We'll refer to a transport-layer packet as a segment.

Network Layer

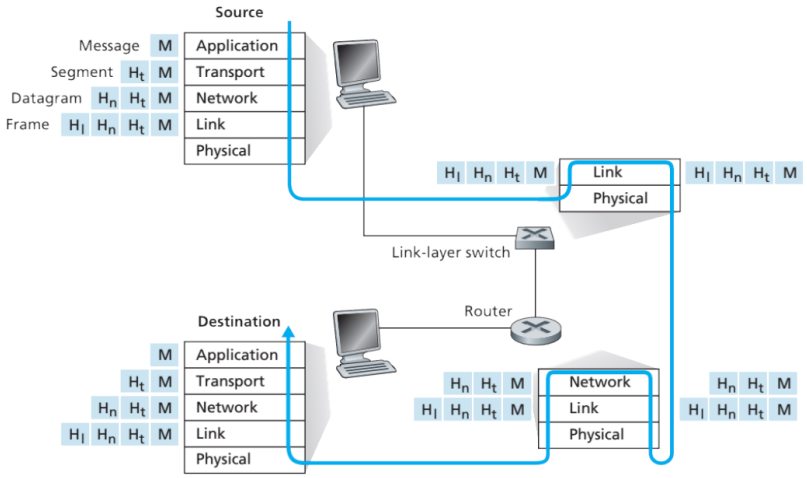
The Internet's network layer is responsible for moving network-layer packets known as datagrams from one host to another. The Internet's network layer includes the celebrated IP Protocol, which defines the fields in the datagram as well as how the end systems and routers act on these fields.

Link Layer

The Internet's network layer routes a datagram through a series of routers between the source and destination. To move a packet from one node (host or router) to the next node in the route, the network layer relies on the services of the link layer. In particular, at each node, the network layer passes the datagram down to the link layer, which delivers the datagram to the next node along the route. At this next node, the link layer passes the datagram up to the network layer. Examples of linklayer protocols include Ethernet, WiFi. As datagrams typically need to traverse several links to travel from source to destination, a datagram may be handled by different link-layer protocols at different links along its route. We'll refer to the linklayer packets as frames.

Physical Layer

While the job of the link layer is to move entire frames from one network element to an adjacent network element, the job of the physical layer is to move the individual bits within the frame from one node to the next. The protocols in this layer are again link dependent and further depend on the actual transmission medium of the link (for example, twisted-pair copper wire, single-mode fiber optics).



[Kurose and Ross] Hosts, routers, and link-layer switches; each contains a different set of layers, reflecting their differences in functionality.

Thus, we see that at each layer, a packet has two types of fields: header fields and a payload field. The payload is typically a packet from the layer above.

Physical Layer

Fourier Analysis

Fourier proved that any reasonably behaved periodic function, $g(t)$ with period T, can be constructed as the sum of a (possibly infinite) number of sines and cosines:

$$g(t) = \frac{1}{2}c + \sum_{n=1}^{\infty} a_n \sin(2\pi nft) + \sum_{n=1}^{\infty} b_n \cos(2\pi nft)$$

where $f = 1/T$ is the fundamental frequency, a_n and b_n are the sine and cosine amplitudes of the n th harmonics (terms), and c is a constant.

The a_n amplitudes can be computed for any given $g(t)$ by multiplying both sides of Eq. by $\sin(2\pi kft)$ and then integrating from 0 to T. since

$$\int_0^T \sin(2\pi kft)\sin(2\pi nft)dt = \begin{cases} 0 \text{ for } k \neq n \\ T/2 \text{ for } k = n \end{cases}$$

only one term of the summation survives: a_n. The b_n summation vanishes completely. Similarly, by multiplying Eq. by $\cos(2\pi kft)$ and integrating between 0 and T, we can derive b_n. By just integrating both sides of the equation as it stands, we can find c. The results of performing these operations are as follows:

$$a_n = \frac{2}{T}\int_0^T g(t)\sin(2\pi nft)dt \quad b_n = \frac{2}{T}\int_0^T g(t)\cos(2\pi nft)dt \quad c = \frac{2}{T}\int_0^T g(t)dt$$

Bandwidth-Limited Signals

The relevance of all of this to data communication is that real channels affect different frequency signals differently. Let us consider a specific example: the transmission of the ASCII character "b" encoded in an 8-bit byte. The bit pattern that is to be transmitted is 01100010. The left-hand part of Fig. (a) shows the voltage output by the transmitting computer. The Fourier analysis of this signal yields the coefficients:

$$a_n = \frac{1}{\pi n}[\cos(\pi n/4) - \cos(3\pi n/4) + \cos(6\pi n/4) - \cos(7\pi n/4)]$$
$$b_n = \frac{1}{\pi n}[\sin(3\pi n/4) - \sin(\pi n/4) + \sin(7\pi n/4) - \sin(6\pi n/4)]$$
$$c = 3/4$$

The root-mean-square amplitudes, $\sqrt{a_n^2 + b_n^2}$, for the first few terms are shown on the right-hand side of Fig. 2-1(a). These values are of interest because their squares are proportional to the energy transmitted at the corresponding frequency. No transmission facility can transmit signals without losing some power in the process. If all the Fourier components were equally diminished, the resulting signal would be reduced in amplitude but not distorted [i.e., it would have the same nice squared-off shape as Fig. 2-1(a)]. Unfortunately, all transmission facilities diminish different Fourier components by different amounts, thus introducing distortion. Usually, for a wire, the amplitudes are transmitted mostly undiminished from 0 up to some frequency f_c [measured in cycles/sec or Hertz (Hz)], with all frequencies above this cutoff frequency attenuated. The width of the frequency range transmitted without being strongly attenuated is called the bandwidth. In practice, the cutoff is not really sharp, so often the quoted bandwidth is from 0 to the frequency at which the received power has fallen by half.

The bandwidth is a physical property of the transmission medium that depends on, for example, the construction, thickness, and length of a wire or fiber. Filters are often used to further limit the bandwidth of a signal. 802.11 wireless channels are allowed to use up to roughly 20 MHz, for example, so 802.11 radios filter the signal bandwidth to this size. As another example, traditional (analog) television channels occupy 6 MHz each, on a wire or over the air. This filtering lets more signals share a given region of spectrum, which improves the overall efficiency of the system. It means that the frequency range for some signals will not start at zero, but this does not matter. The bandwidth is still the width of the band of frequencies that are passed, and the information that can be carried de- pends only on this width and not on the starting and ending frequencies. Signals that run from 0 up to a maximum frequency are called baseband signals. Signals that are shifted to occupy a higher range of frequencies, as is the case for all wireless transmissions, are called passband signals. So bandwidth = baseband + passband.

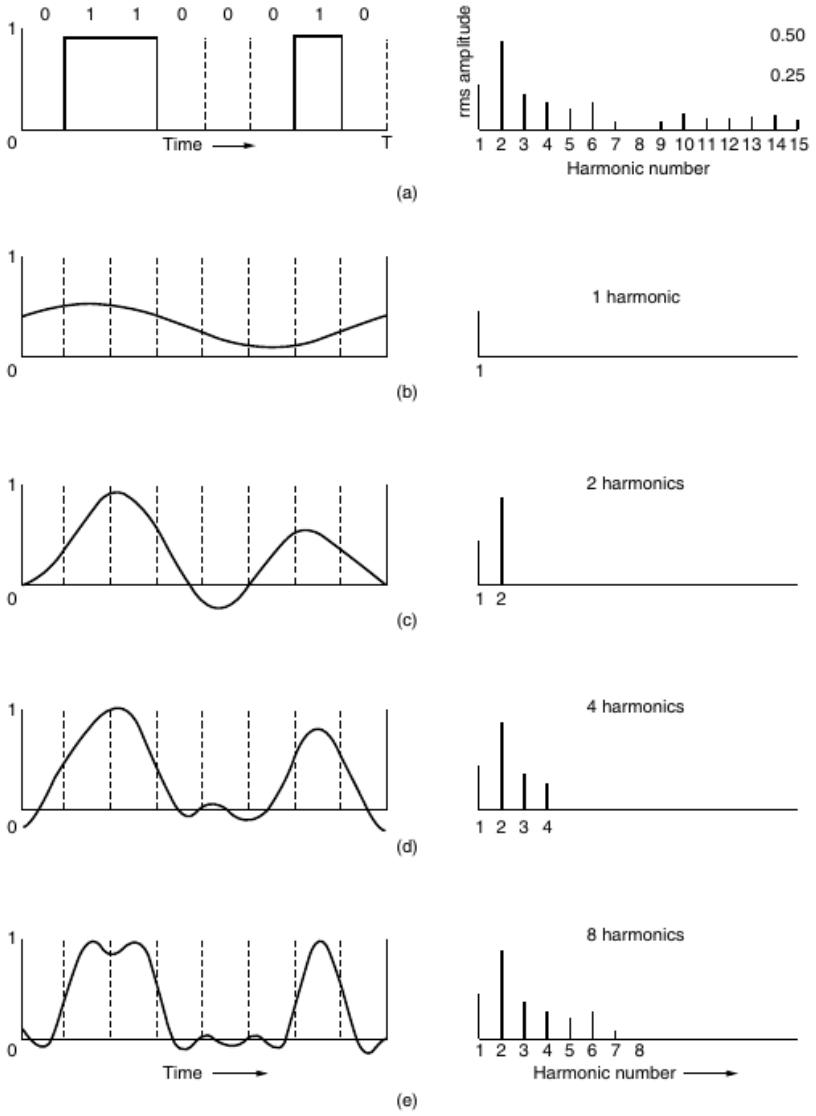


Figure 2-1. [Tanenbaum] (a) A binary signal and its root-mean-square Fourier amplitudes. (b)–(e) Successive approximations to the original signal.

Now let us consider how the signal of Fig. 2-1(a) would look if the bandwidth were so low that only the lowest frequencies were transmitted [i.e., if the function were being approximated by the first few terms of Eq. (2-1)]. Figure 2-1(b) shows the signal that results from a channel that allows only the first harmonic (the fundamental, f) to pass through. Similarly, Fig. 2-1(c)–(e) show the spectra and reconstructed functions for higher-bandwidth channels. For digital transmission, the goal is to receive a signal with just enough fidelity to

reconstruct the sequence of bits that was sent. We can already do this easily in Fig. 2-1(e), so it is wasteful to use more harmonics to receive a more accurate replica.

Given a bit rate of b bits/sec, the time required to send the 8 bits in our example 1 bit at a time is 8/b sec, so the frequency of the first harmonic of this signal is b /8 Hz. An ordinary telephone line, often called a voice-grade line, has an artificially introduced cutoff frequency just above 3000 Hz. The presence of this restriction means that the number of the highest harmonic passed through is roughly 3000/(b/8), or 24,000/b (the cutoff is not sharp). For some data rates, the numbers work out as shown in Fig. 2-2. From these numbers, it is clear that trying to send at 9600 bps over a voice-grade telephone line will transform Fig. 2-1(a) into something looking like Fig. 2-1(c), making accurate reception of the original binary bit stream tricky. It should be obvious that at data rates much higher than 38.4 kbps, there is no hope at all for binary signals, even if the transmission facility is completely noiseless. In other words, limiting the bandwidth limits the data rate, even for perfect channels. However, coding schemes that make use of several voltage levels do exist and can achieve higher data rates. We will discuss these later in this chapter.

Bps (b)	T (msec) (8/b)	First Harmonic (Hz) (b/8)	# Harmonics sent
300	26.67	37.5	80
600	13.33	75	40
1200	6.67	150	20
2400	3.33	300	10
4800	1.67	600	5
9600	0.83	1200	2
19200	0.42	2400	1
38400	0.21	4800	0

Figure 2-2. Relation between data rate and harmonics for our example.

The Maximum Data Rate of a Channel

Nyquist proved that if an arbitrary signal has been run through a low-pass filter of bandwidth B, the filtered signal can be completely reconstructed by making only 2B (exact) samples per second. Sampling the line faster than 2B times per second is pointless because the higher-frequency components that such sampling could recover have already been filtered out. If the signal consists of V discrete levels, Nyquist's theorem states: maximum data rate = $2B log_2 V$ bits/sec (2-2) For example, a noiseless 3-kHz channel cannot transmit binary (i.e., two-level) signals at a rate exceeding 6000 bps. So far we have considered only noiseless channels. If random noise is present, the situation deteriorates rapidly. And there is always random (thermal) noise present due to the motion

of the molecules in the system. The amount of thermal noise present is measured by the ratio of the signal power to the noise power, called the SNR (Signal-to-Noise Ratio). If we denote the signal power by S and the noise power by N, the signal-to-noise ratio is S/N. Usually, the ratio is expressed on a log scale as the quantity $10 \log_{10} S/N$ because it can vary over a tremendous range. The units of this log scale are called decibels (dB), with "deci" meaning 10 and "bel" chosen to honor Alexander Graham Bell, who invented the telephone. An S /N ratio of 10 is 10 dB, a ratio of 100 is 20 dB, a ratio of 1000 is 30 dB, and so on. The manufacturers of stereo amplifiers often characterize the bandwidth (frequency range) over which their products are linear by giving the 3dB frequency on each end. These are the points at which the amplification factor has been approximately halved (because $10 \log_{10} 0.5 \approx -3$). Shannon's major result is that the maximum data rate or capacity of a noisy channel whose bandwidth is B Hz and whose signal-to-noise ratio is S/N, is given by: maximum number of bits/sec $= B \log_2(1 + S/N)$ (2-3)

This tells us the best capacities that real channels can have. For example, ADSL (Asymmetric Digital Subscriber Line), which provides Internet access over normal telephone lines, uses a bandwidth of around 1 MHz. The SNR depends strongly on the distance of the home from the telephone exchange, and an SNR of around 40 dB for short lines of 1 to 2 km is very good. With these characteristics, the channel can never transmit much more than 13 Mbps ($4log_2(10) \approx 13$), no matter how many or how few signal levels are used and no matter how often or how infrequently samples are taken. In practice, ADSL is specified up to 12 Mbps, though users often see lower rates. This data rate is actually very good, with over 60 years of communications techniques having greatly reduced the gap between the Shannon capacity and the capacity of real systems.

Digital Modulation And Multiplexing

Wires and wireless channels carry analog signals such as continuously varying voltage, light intensity, or sound intensity. To send digital information, we must devise analog signals to represent bits. The process of converting between bits and signals that represent them is called digital modulation.

We will start with schemes that directly convert bits into a signal. These schemes result in baseband transmission, in which the signal occupies frequencies from zero up to a maximum that depends on the signaling rate. It is common for wires. Then we will consider schemes that regulate the amplitude, phase, or frequency of a carrier signal to convey bits. These schemes result in passband transmission, in which the signal occupies a band of frequencies around the frequency of the carrier signal. It is common for wireless and optical channels for which the signals must reside in a given frequency band. Channels are often shared by multiple signals. After all, it is much more convenient to use a single wire to carry several signals than to install a wire for every signal. This kind of

sharing is called multiplexing. It can be accomplished in several different ways. We will present methods for time, frequency, and code division multiplexing.

Baseband Transmission

The most straightforward form of digital modulation is to use a positive voltage to represent a 1 and a negative voltage to represent a 0. This scheme is called NRZ (Non-Return-to-Zero). An example is shown in Fig. 2-20(b).

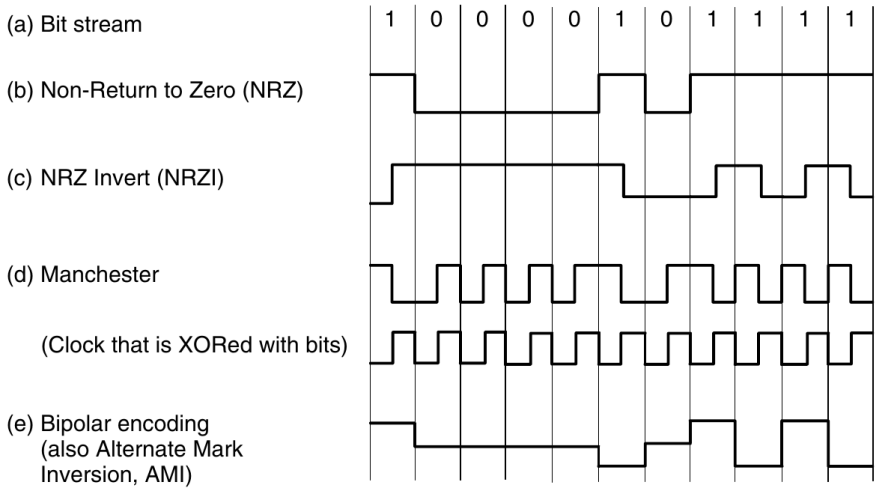


Figure 2-3. [Tanenbaum] Line codes: (a) Bits, (b) NRZ, (c) NRZI, (d) Manchester, (e) Bipolar or AMI.

Once sent, the NRZ signal propagates down the wire. At the other end, the receiver converts it into bits by sampling the signal at regular intervals of time.

This signal will not look exactly like the signal that was sent. It will be attenuated and distorted by the channel and noise at the receiver. To decode the bits, the receiver maps the signal samples to the closest symbols. For NRZ, a positive voltage will be taken to indicate that a 1 was sent and a negative voltage will be taken to indicate that a 0 was sent.

One strategy for using limited bandwidth more efficiently is to use more than two signaling levels. By using four voltages, for instance, we can send 2 bits at once as a single symbol. This design will work as long as the signal at the receiver is sufficiently strong to distinguish the four levels. The rate at which the signal changes is then half the bit rate, so the needed bandwidth has been reduced.

We call the rate at which the signal changes the symbol rate to distinguish it from the bit rate. The bit rate is the symbol rate multiplied by the number of bits per symbol.

Clock Recovery For all schemes that encode bits into symbols, the receiver must know when one symbol ends and the next symbol begins to correctly decode the bits. With NRZ, in which the symbols are simply voltage levels, a long run of 0s or 1s leaves the signal unchanged. After a while it is hard to tell the bits apart, as 15 zeros look much like 16 zeros unless you have a very accurate clock.

Accurate clocks would help with this problem, but they are an expensive solution for commodity equipment.

One strategy is to send a separate clock signal to the receiver. Another clock line is no big deal for computer buses or short cables in which there are many lines in parallel, but it is wasteful for most network links since if we had another line to send a signal we could use it to send data. A clever trick here is to mix the clock signal with the data signal by XORing them together so that no extra line is needed. The results are shown in Fig. 2-2(d). The clock makes a clock transition in every bit time, so it runs at twice the bit rate. This scheme is called Manchester encoding. The downside of Manchester encoding is that it requires twice as much band-width as NRZ because of the clock. A different strategy is based on the idea that we should code the data to ensure that there are enough transitions in the signal. Consider that NRZ will have clock recovery problems only for long runs of 0s and 1s. If there are frequent transitions, it will be easy for the receiver to stay synchronized with the incoming stream of symbols.

As a step in the right direction, we can simplify the situation by coding a 1 as a transition and a 0 as no transition, or vice versa. This coding is called NRZI (Non-Return-to-Zero Inverted), a twist on NRZ. An example is shown in Fig. 2-2(c). With it, long runs of 1s do not cause a problem. Of course, long runs of 0s still cause a problem that we must fix. To really fix the problem we can break up runs of 0s by mapping small groups of bits to be transmitted so that groups with successive 0s are mapped to slightly longer patterns that do not have too many consecutive 0s. A well-known code to do this is called 4B/5B. Every 4 bits is mapped into a5-bit pattern with a fixed translation table. The five bit patterns are chosen so that there will never be a run of more than three consecutive 0s. This scheme adds 25% overhead, which is better than the 100% overhead of Manchester encoding. Since there are 16 input combinations and 32 output combinations, some of the output combinations are not used. Putting aside the combinations with too many successive 0s, there are still some codes left. As a bonus, we can use these nondata codes to represent physical layer control signals. For example, in some uses “11111” represents an idle line and “11000” represents the start of a frame.

Data (4B)	Codeword (5B)	Data (4B)	Codeword (5B)
0000	11110	1000	10010
0001	01001	1001	10011
0010	10100	1010	10110
0011	10101	1011	10111
0100	01010	1100	11010
0101	01011	1101	11011
0110	01110	1110	11100
0111	01111	1111	11101

Figure 2-4. [Tanenbaum] 4B/5B mapping.

FDM

FDM (Frequency Division Multiplexing) takes advantage of passband transmission to share a channel. It divides the spectrum into frequency bands, with each user having exclusive possession of some band in which to send their signal.

In Fig. 2-25 we show three voice-grade telephone channels multiplexed using FDM. Filters limit the usable bandwidth to about 3100 Hz per voice-grade channel. When many channels are multiplexed together, 4000 Hz is allocated per channel. The excess is called a guard band. It keeps the channels well separated. First the voice channels are raised in frequency, each by a different amount. Then they can be combined because no two channels now occupy the same portion of the spectrum. Notice that even though there are gaps between the channels thanks to the guard bands, there is some overlap between adjacent channels. The overlap is there because real filters do not have ideal sharp edges. This means that a strong spike at the edge of one channel will be felt in the adjacent one as nonthermal noise.

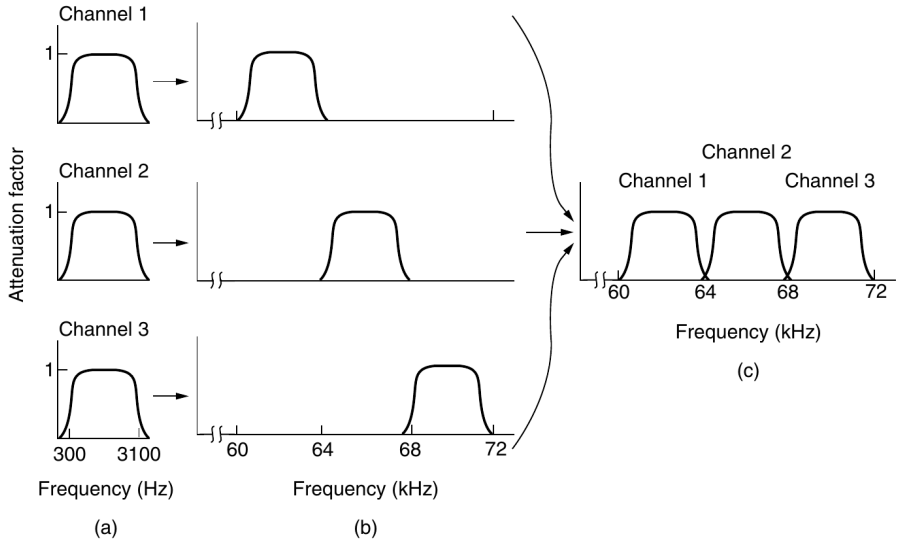


Figure 2-5: [Tanenbaum] Frequency division multiplexing. (a) The original bandwidths. (b) The bandwidths raised in frequency. (c) The multiplexed channel.

TDM

Here, the users take turns (in a round-robin fashion), each one periodically getting the entire bandwidth for a little burst of time. An example of three streams being multiplexed with TDM is shown in Fig. 2-6. Bits from each input stream are taken in a fixed time slot and output to the aggregate stream. This stream runs at the sum rate of the individual streams. For this to work, the streams must be synchronized in time. Small intervals of guard time analogous to a frequency guard band may be added to accommodate small timing variations.

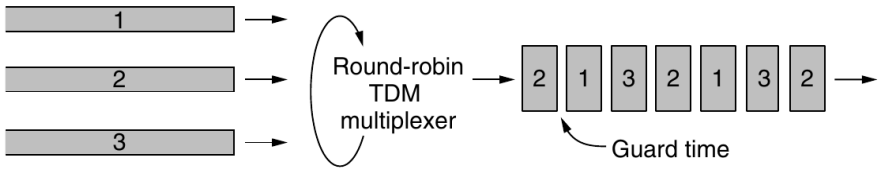


Figure 2-6: [Tanenbaum] TDM

CDM

There is a third kind of multiplexing that works in a completely different way than FDM and TDM. CDM (Code Division Multiplexing) is a form of spread

spectrum communication in which a narrowband signal is spread out over a wider frequency band. This can make it more tolerant of interference, as well as allowing multiple signals from different users to share the same frequency band. Because code division multiplexing is mostly used for the latter purpose it is commonly called CDMA (Code Division Multiple Access). CDMA allows each station to transmit over the entire frequency spectrum all the time.

In CDMA, each bit time is subdivided into m short intervals called chips. Typically, there are 64 or 128 chips per bit, but in the example given here we will use 8 chips/bit for simplicity. Each station is assigned a unique m-bit code called a chip sequence. It is convenient to use a bipolar notation to write these codes as sequences of -1 and $+1$. We will show chip sequences in parentheses.

To transmit a 1 bit, a station sends its chip sequence. To transmit a 0 bit, it sends the negation of its chip sequence. No other patterns are permitted. Thus, for m = 8, if station A is assigned the chip sequence (-1 -1 -1 $+1$ $+1$ -1 $+1$ $+1$), it can send a 1 bit by transmiting the chip sequence and a 0 by transmitting ($+1$ $+1$ $+1$ -1 -1 $+1$ -1 -1).

Increasing the amount of information to be sent from b bits/sec to mb chips/sec for each station means that the bandwidth needed for CDMA is greater by a factor of m than the bandwidth needed for a station not using CDMA (assuming no changes in the modulation or encoding techniques). If we have a 1-MHz band available for 100 stations, with FDM each one would have 10 kHz and could send at 10 kbps (assuming 1 bit per Hz). With CDMA, each station uses the full 1 MHz, so the chip rate is 100 chips per bit to spread the station's bit rate of 10 kbps across the channel.

Each station has its own unique chip sequence. Let us use the symbol S to indicate the m -chip vector for station S , and $\overline{S}$ for its negation. All chip sequences are pairwise orthogonal, by which we mean that the normalized inner product of any two distinct chip sequences, $\mathbf{S}$ and $\mathbf{T}$ (written as $\mathbf{S} \bullet \mathbf{T}$), is 0. It is known how to generate such orthogonal chip sequences using a method known as Walsh codes. In mathematical terms, orthogonality of the chip sequences can be expressed as follows:

$\mathbf{S} \bullet \mathbf{T} \equiv \frac{1}{m} \sum_{i=1}^{m} S_i T_i = 0$

Note that if $\mathbf{S} \bullet \mathbf{T} = 0$, then $\mathbf{S} \bullet \overline{\mathbf{T}}$ is also 0. The normalized inner product of any chip sequence with itself is 1 :

$\mathbf{S} \bullet \mathbf{S} = \frac{1}{m} \sum_{i=1}^{m} S_i S_i = \frac{1}{m} \sum_{i=1}^{m} S_i^2 = \frac{1}{m} \sum_{i=1}^{m} (\pm 1)^2 = 1$

Also note that $\mathbf{S} \bullet \overline{\mathbf{S}} = -1$

The combined signal is

$$Y = \sum_i b_i \cdot S_i + (1 - b_i) \cdot \overline{S}_i$$

where $b_i \in \{0, 1\}$. To recover station j' s signal, the receiver will just take the inner product of Y with S_j . i.e.,

$$\begin{aligned} Y \bullet S_j &= \sum_i b_i \cdot S_i \bullet S_j + (1 - b_i) \cdot \overline{S_i} \bullet S_j \\ &= b_j + \left(1 - b_j\right) \cdot -1 = 2b_j - 1 \end{aligned}$$

Equivalently, we have

$$b_j = \left(Y \bullet S_j + 1\right)/2$$

Link Layer

- We will refer to any device that runs a link-layer protocol as node.
- The communication channels that connect adjacent nodes along the communication path as links.
- In order to transfer data transferred from source node to destination node, it must be moved over each of the individual links in the end-to-end path.
- Transmitting nodes encapsulate datagram in a link-layer frame and transmits the frame into the link.

Link Layer Services

- **Framing**: A frame consists of a data field, in which the network-layer datagram is inserted, and a number of header fields.
- **Reliable delivery**: When a link-layer protocol provides reliable delivery service, it guarantees to move each network-layer datagram across the link without error.
- **Link access**: A medium access control (MAC) protocol specifies the rules by which a frame is transmitted onto the link. For point-to-point links, the MAC protocol is simple. The more interesting case is when multiple nodes share a single broadcast link—the so-called multiple access problem.
- **Error detection and correction**: The link-layer hardware in a receiving node can incorrectly decide that a bit in a frame is zero when it was transmitted as a one, and vice versa

Framing

- The bit stream received by the data link layer is not guaranteed to be error free.
- It is up to the data link layer to detect and, if necessary, correct errors.
- Data link layer to break up the bit stream into discrete frames, compute a short token called a checksum for each frame, and include the checksum in the frame when it is transmitted.
- When a frame arrives at the destination, the checksum is recomputed. If the newly computed checksum is different from the one contained in the

frame, the data link layer knows that an error has occurred and takes steps to deal with it.

- Four methods for Framing:
  - Byte count.
  - Flag bytes with byte stuffing.
  - Flag bytes with bit stuffing.
  - Physical layer coding violations.

Byte count

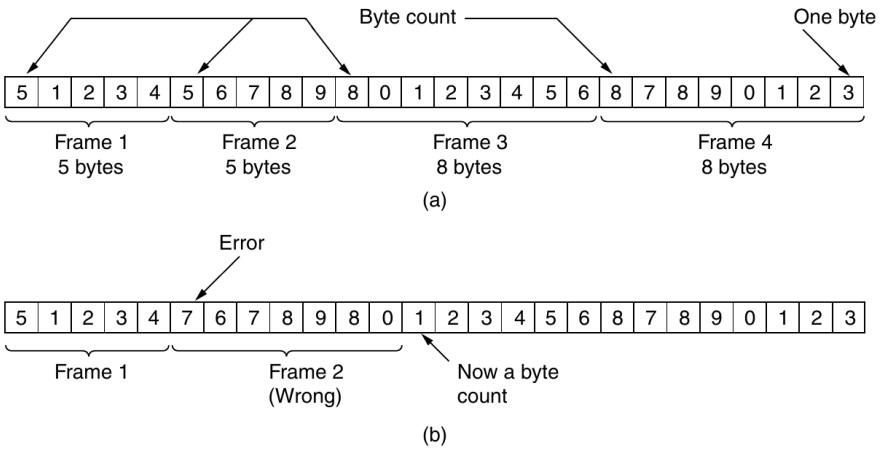


Figure [Tanenbaum] A byte stream (a) Without errors. (b) With one error

Flag bytes with byte stuffing

Two consecutive flag bytes indicate the end of one frame and the start of the next. Thus, if the receiver ever loses synchronization it can just search for two flag bytes to find the end of the current frame and the start of the next frame

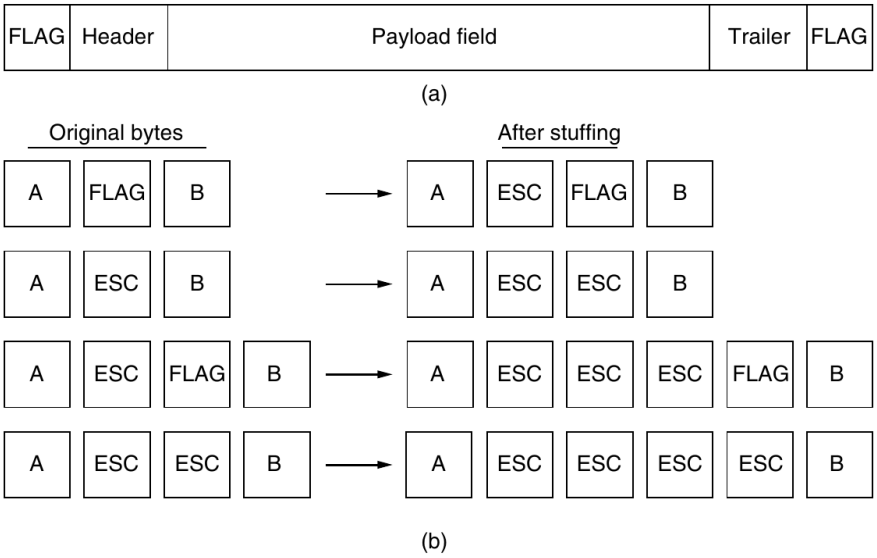


Figure [Tanenbaum] (a) A frame delimited by flag bytes. (b) Four examples of byte sequences before and after byte stuffing.

$2B$ overhead.

Flag bytes with bit stuffing

Each frame begins and ends with a special bit pattern, 01111110 or 0x7E in hexadecimal. This pattern is a flag byte.

Whenever the sender's data link layer encounters five consecutive 1s in the data, it automatically stuffs a 0 bit into the outgoing bit stream.

When the receiver sees five consecutive incoming 1 bits, followed by a 0 bit, it automatically destuffs (i.e., deletes) the 0 bit.

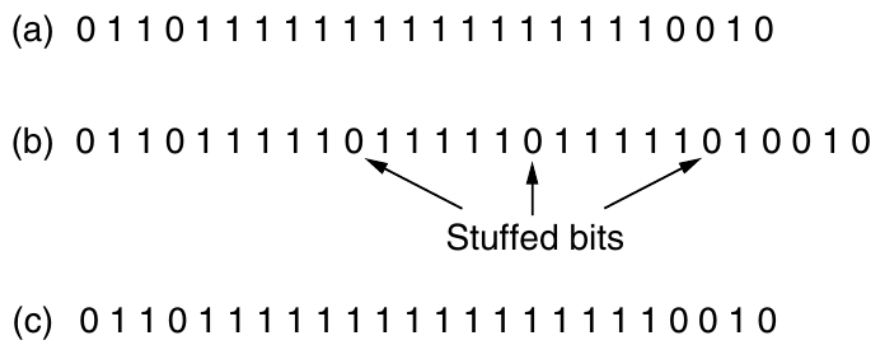


Figure [Tanenbaum] (a) The original data. (b) The data as they appear on the line. (c) The data as they are stored in the receiver's memory after destuffing

$B/5$ overhead. (But hard disks prefer byte reads and writes)

Physical layer coding violations

- Encoding of bits as signals often includes redundancy to help the receiver.
- For example, in the 4B/5B line code 4 data bits are mapped to 5 signal bits to ensure sufficient bit transitions. We can use some reserved signals to indicate the start and end of frames.
- We are using "coding violations" to delimit frames.
- It is easy to find the start and end of frames and there is no need to stuff the data.

Error Control

Having solved the problem of marking the start and end of each frame, we come to the next problem: how to make sure all frames are eventually delivered to the network layer at the destination and in the proper order.

For which we have seen, acknowledgements, timer, etc.

Flow Control

Another important design issue that occurs in the data link layer (and higher layers as well) is what to do with a sender that systematically wants to transmit frames faster than the receiver can accept them. This situation can occur when the sender is running on a fast, powerful computer and the receiver is running on a slow, low-end machine.

Two approaches are commonly used. In the first one, **feedback-based flow control**, the receiver sends back information to the sender giving it permission to send more data, or at least telling the sender how the receiver is doing. In the second one, **rate-based flow control**, the protocol has a built-in mechanism that limits the rate at which senders may transmit data, without using feedback from the receiver.

Error detection and correction

- Transmission errors are unavoidable. We need to develop techniques to deal with them.

- One strategy is to include enough redundant information to enable the receiver to deduce what the transmitted data must have been — Forward Error Correction (FEC)
- The other is to include only enough redundancy to allow the receiver to deduce that an error has occurred and have it request a retransmission.
- When noise is unavoidable like in wireless communication, FEC is must and when there is low probability of error then detection is better.
- A frame consists of m data (i.e., message) bits and r redundant (i.e. check) bits.
- Block code: the r check bits are computed solely as a function of the m data bits with which they are associated
- Systematic code: the m data bits are sent directly, along with the check bits, rather than being encoded themselves before they are sent.
- Linear code: the r check bits are computed as a linear function of the m data bits. Exclusive OR (XOR) or modulo 2 addition is a popular choice.
- The codes we will look at are linear, systematic block codes unless otherwise noted.
- Let the total length of a block be n (i.e., $n = m+r$). We will describe this as an (n, m) code. An n-bit unit containing data and check bits is referred to as an n-bit codeword. The code rate is the fraction of the codeword that carries non-redundant information i.e., m/n.

Hamming Codes

- Given two code words 10001001 and 10110001 — is to possible to determine how many corresponding bits differ?
- This difference is called the Hamming distance
- If two code words are a Hamming distance of d apart, d single-bit errors are needed to convert one into the other.
- In most data transmission applications, all 2^m possible data messages are legal, but due to the way the check bits are computed, not all of the 2^n possible codewords are used. In fact, when there are r check bits, only the small fraction of $2^m/2^n$ or $1/2^r$ of the possible messages will be legal codewords.
- Given the algorithm for computing the check bits, it is possible to construct a complete list of the legal codewords, and from this list to find the two codewords with the smallest Hamming distance. This distance is the Hamming distance of the complete code.
- To reliably detect d errors, you need a distance $d + 1$ code because with such a code there is no way that d single-bit errors can change a valid

codeword into another valid codeword. When the receiver sees an illegal codeword, it can tell that a transmission error has occurred.

- Similarly, to correct d errors, you need a distance $2d+1$ code because that way the legal codewords are so far apart that even with d changes the original codeword is still closer than any other codeword. This means the original codeword can be uniquely determined based on the assumption that a larger number of errors are less likely.
- We want to design a code with m message bits and r check bits that will allow all single errors to be corrected.
- Each of the 2^m legal messages has n illegal codewords at a distance of 1 from it.
- Each of the 2^m legal messages requires $n+1$ bit patterns dedicated to it.
- We get the requirement that $2^m(n+1) \leq 2^n \rightarrow (m+r+1) \leq 2^r$.
- To correct 2 bit errors $\rightarrow 2^m(\binom{n}{2}+1) \leq 2^n$
- (For 1 bit error correction, consider $|m_i| = 7$) Place check bits at index power of two because then $d(C(m_1), C(m_2)) \geq 3$ whenever $d(m_1, m_2) \geq 1$ as index of bit difference will not be a power of 2.

Convolutional Code

- Not a block code.
- In a convolutional code, an encoder processes a sequence of input bits and generates a sequence of output bits. There is no natural message size or encoding boundary as in a block code.
- The output depends on the current and previous input bits. That is, the encoder has memory. The number of previous bits on which the output depends is called the **constraint length** (includes current bit) of the code.
- Convolutional codes are specified in terms of their rate ($\frac{\text{\# Input bits}}{\text{\# Output Bits produced}}$) and constraint length.

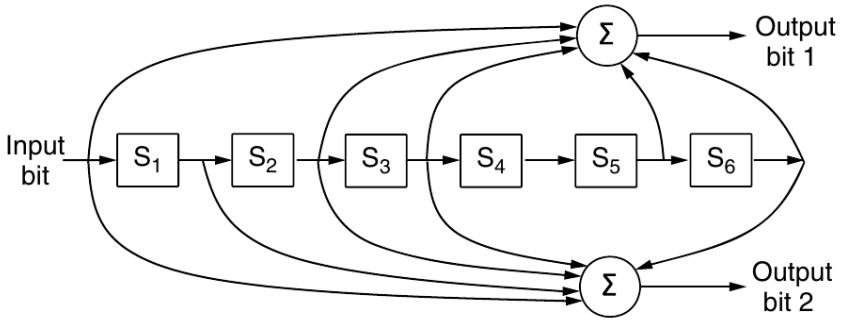


Figure: [Tanenbaum] The NASA binary convolutional code used in 802.11.

In figure, each input bit on the left-hand side produces two output bits on the right-hand side that are XOR sums of the input and internal state. Since it deals with bits and performs linear operations, this is a binary, linear convolutional

code. Since 1 input bit produces 2 output bits, the code rate is 1/2. It is not systematic since none of the output bits is simply the input bit.

The internal state is kept in six memory registers. Each time another bit is input the values in the registers are shifted to the right. For example, if 111 is input and the initial state is all zeros, the internal state, written left to right, will become 100000, 110000, and 111000 after the first, second, and third bits have been input. The output bits will be 11, followed by 10, and then 01. It takes seven shifts to flush an input completely so that it does not affect the output. The constraint length of this code is thus k = 7.

A convolutional code is decoded by finding the sequence of input bits that is most likely to have produced the observed sequence of output bits (which includes any errors). For small values of k, this is done with a widely used algorithm developed by Viterbi (Forney, 1973). The algorithm walks the observed sequence, keeping for each step and for each possible internal state the input sequence that would have produced the observed sequence with the fewest errors. The input sequence requiring the fewest errors at the end is the most likely message. Convolutional codes have been popular in practice because it is easy to factor the uncertainty of a bit being a 0 or a 1 into the decoding.

Reed-Solomon Code

- Are linear block codes, and they are often systematic too.
- Unlike Hamming codes, which operate on individual bits, Reed-Solomon codes operate on m bit symbols.
- Reed-Solomon codes are based on the fact that every n degree polynomial is uniquely determined by n + 1 points. For example, a line having the form ax + b is determined by two points. Extra points on the same line are redundant, which is helpful for error correction. Imagine that we have two data points that represent a line and we send those two data points plus two check points chosen to lie on the same line. If one of the points is received in error, we can still recover the data points by fitting a line to the received points. Three of the points will lie on the line, and one point, the one in error, will not. By finding the line we have corrected the error.
- Reed-Solomon codes are actually defined as polynomials that operate over finite fields, but they work in a similar manner. For m bit symbols, the codewords are $2^m - 1$ symbols long. A popular choice is to make m = 8 so that symbols are bytes. A codeword is then 255 bytes long. The (255, 233) code is widely used; it adds 32 redundant symbols to 233 data symbols. Decoding with error correction is done with an algorithm developed by Berlekamp and Massey that can efficiently perform the fitting task for moderate-length codes
- Reed-Solomon codes are widely used in practice because of their strong error-correction properties, particularly for burst errors. Because they are

based on m bit symbols, a single-bit error and an m-bit burst error are both treated simply as one symbol error.
- When $2t$ redundant symbols are added, a Reed-Solomon code is able to correct up to t errors in any of the transmitted symbols. This means, for example, that the (255, 233) code, which has 32 redundant symbols, can correct up to 16 symbol errors. Since the symbols may be consecutive and they are each 8 bits, an error burst of up to 128 bits can be corrected.

Error Detecting Codes

We will examine three different error-detecting codes. They are all linear, systematic block codes.

To see how they can be more efficient than error-correcting codes, consider:-

Parity

- Consider a case where a single parity bit is appended to the data.
- The parity bit is chosen so that the number of 1 bits in the codeword is even (or odd). For example, when 1011010 is sent in even parity, a bit is added to the end to make it 10110100. With odd parity 1011010 becomes 10110101.
- A code with a single parity bit has a distance of 2, since any single-bit error produces a codeword with the wrong parity. This means that it can detect single-bit errors.
- One difficulty with this scheme is that a single parity bit can only reliably detect a single-bit error in the block. If the block is badly garbled by a long burst error, the probability that the error will be detected is only 0.5.
- The odds can be improved considerably if each block to be sent is regarded as a rectangular matrix n bits wide and k bits high. Now, if we compute and send one parity bit for each row, up to k bit errors will be reliably detected as long as there is at most one error per row. However, there is something else we can do that provides better protection against burst errors: we can compute the parity bits over the data in a different order than the order in which the data bits are transmitted. Doing so is called interleaving. In this case, we will compute a parity bit for each of the n columns and send all the data bits as k rows, sending the rows from top to bottom and the bits in each row from left to right in the usual manner. At the last row, we send the n parity bits. This transmission order is shown in below Fig for n = 7 and k = 7.

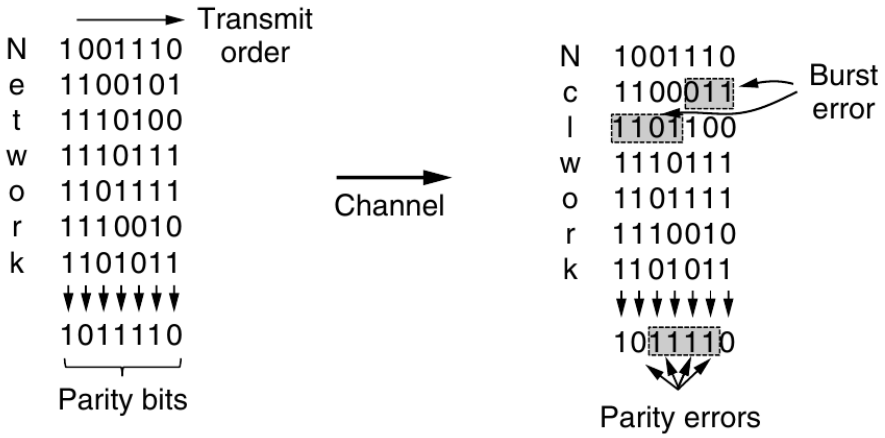


Figure: [Tanenbaum] Interleaving of parity bits to detect a burst error. (A burst error does not imply that all the bits are wrong; it just implies that at least the first and last are wrong.)

Interleaving is a general technique to convert a code that detects (or corrects) isolated errors into a code that detects (or corrects) burst errors. This method uses n parity bits on blocks of kn data bits to detect a single burst error of length n or less.

A burst of length $n+1$ will pass undetected, however, if the first bit is inverted, the last bit is inverted, and all the other bits are correct. If the block is badly garbled by a long burst or by multiple shorter bursts, the probability that any of the n columns will have the correct parity by accident is 0.5, so the probability of a bad block being accepted when it should not be is 2^{-n} .

Checksum

Checksums are usually based on a running sum of the data bits of the message. The checksum is usually placed at the end of the message, as the complement of the sum function. This way, errors may be detected by summing the entire received codeword, both data bits and checksum. If the result comes out to be zero, no error has been detected.

One example of a checksum is the 16-bit Internet checksum used on all Internet packets as part of the IP protocol (Braden et al., 1988). This checksum is a sum of the message bits divided into 16-bit words. Because this method operates on words rather than on bits, as in parity, errors that leave the parity unchanged can still alter the sum and be detected. For example, if the lowest order bit in two different words is flipped from a 0 to a 1, a parity check across these bits would fail to detect an error. However, two 1s will be added to the 16-bit checksum to produce a different result. The error can then be detected.

Cyclic Redundancy Check (CRC)

- Aka Polynomial codes are based upon treating bit strings as representations of polynomials with coefficients of 0 and 1 only. A k-bit frame is regarded as the coefficient list for a polynomial with k terms, ranging from x^{k-1} to x^0. The high-order (leftmost) bit is the coefficient of x^{k-1} and so on.
- Polynomial arithmetic is done modulo 2, according to the rules of algebraic field theory. It does not have carries for addition or borrows for subtraction. Both addition and subtraction are identical to exclusive OR.
- Long division is carried out in exactly the same way as it is in binary except that the subtraction is again done modulo 2.
- When the polynomial code method is employed, the sender and receiver must agree upon a generator polynomial, $G(x)$, in advance. Both the high- and low- order bits of the generator must be 1. To compute the CRC for some frame with m bits corresponding to the polynomial $M(x)$, the frame must be longer than the generator polynomial. The idea is to append a CRC to the end of the frame in such a way that the polynomial represented by the checksummed frame is divisible by $G(x)$. When the receiver gets the checksummed frame, it tries dividing it by $G(x)$. If there is a remainder, there has been a transmission error.
- The algorithm for computing the CRC is as follows:
  1. Let r be the degree of $G(x)$. Append r zero bits to the low-order end of the frame so it now contains $m + r$ bits and corresponds to the polynomial $x^r M(x)$.
  2. Divide the bit string corresponding to $G(x)$ into the bit string corresrponding to $x^r M(x)$, using modulo 2 division.
  3. Subtract the remainder (which is always r or fewer bits) from the bit string corresponding to $x^r M(x)$ using modulo 2 subtraction. The result is the checksummed frame to be transmitted. Call its polynomial $T(x)$.

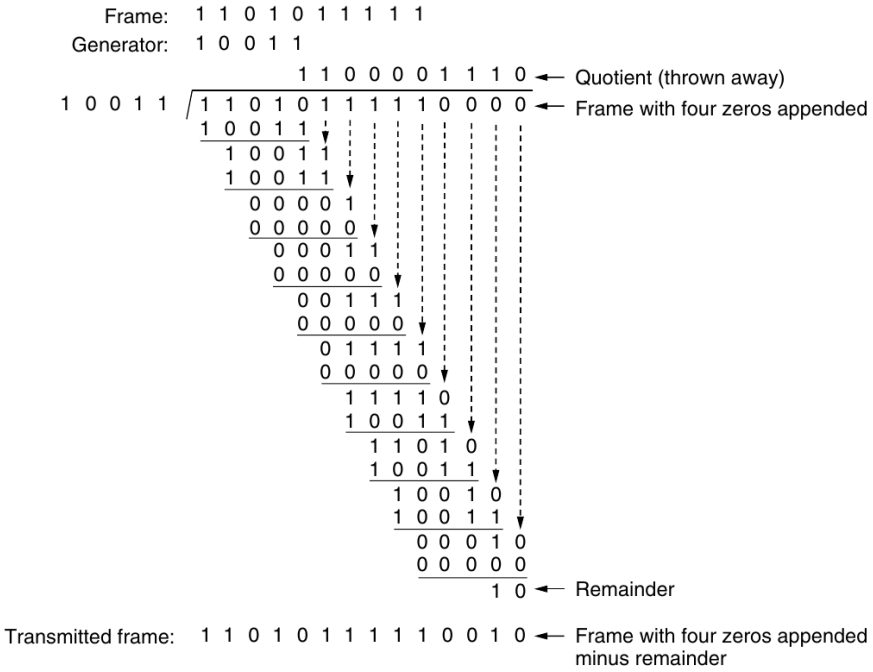


Figure: [Tanenbaum] Example Calculation of CRC

- Note that we require that lower order bit be 1 for G(x). If not the CRC checksum calculated for any M(x) will always have 0 in as its last bit and thus would not contribute to error detection in the final (message + checksum).
- $x^r M(x)$ instead of $M(x)$ as o/w we will be modifying the message itself.
- Interestingly, no polynomial with an odd number of terms has x + 1 as a factor in the modulo 2 system. Proof here.
- What about burst errors? A burst error of length k can be represented by $x^i(x^{k-1} + \cdots + 1)$. Can detect burst errors of length $\leq r$, where r is the degree of $G(x)$.

Link layer protocols

- A seq_nr is a small integer used to number the frames so that we can tell them apart. These sequence numbers run from 0 up to and including MAX_SEQ, which is defined in each protocol needing it.
- See slides (Lec 10).

Sliding Window Protocols

In the previous protocols, data frames were transmitted in one direction only. In most practical situations, there is a need to transmit data in both directions. One way of achieving full-duplex data transmission is to run two instances of one of the previous protocols, each using a separate link for simplex data traffic (in different directions). Each link is then comprised of a "forward" channel (for data) and a "reverse" channel (for acknowledgements). In both cases the capacity of the reverse channel is almost entirely wasted. A better idea is to use the same link for data in both directions. After all, in protocols 2 and 3 it was already being used to transmit frames both ways, and the reverse channel normally has the same capacity as the forward channel. In this model the data frames from A to B are intermixed with the acknowledgement frames from A to B. By looking at the kind field in the header of an incoming frame, the receiver can tell whether the frame is data or an acknowledgement. Although interleaving data and control frames on the same link is a big im- provement over having two separate physical links, yet another improvement is possible. When a data frame arrives, instead of immediately sending a separate control frame, the receiver restrains itself and waits until the network layer passes it the next packet. The acknowledgement is attached to the outgoing data frame (using the ack field in the frame header). In effect, the acknowledgement gets a free ride on the next outgoing data frame. The technique of temporarily delaying outgoing acknowledgements so that they can be hooked onto the next outgoing data frame is known as piggybacking. If a new packet arrives quickly, the acknowledgement is piggybacked onto it. Otherwise, if no new packet has arrived by the end of this time period, the data link layer just sends a separate acknowledgement frame. The next three protocols are bidirectional protocols that belong to a class cal- led sliding window protocols. In these, as in all sliding window protocols, each outbound frame contains a sequence number, ranging from 0 up to some maximum. The maximum is usually 2n − 1 so the se- quence number fits exactly in an n-bit field. The essence of all sliding window protocols is that at any instant of time, the sender maintains a set of sequence numbers corresponding to frames it is permit- ted to send. These frames are said to fall within the sending window. Similarly, the receiver also maintains a receiving window corresponding to the set of frames it is permitted to accept. The sender's window and the receiver's window need not have the same lower and upper limits or even have the same size. In some protocols they are fixed in size, but in others they can grow or shrink over the course of time as frames are sent and received.

- The sequence numbers within the sender's window represent frames that have been sent or can be sent but are as yet not acknowledged. Whenever a new packet arrives from the network layer, it is given the next highest sequence number, and the upper edge of the window is advanced by one. When an acknowledgement comes in, the lower edge is advanced by one. In this way the window continuously maintains a list of unacknowledged

frames.

- Since frames currently within the sender's window may ultimately be lost or damaged in transit, the sender must keep all of these frames in its memory for possible retransmission. Thus, if the maximum window size is n, the sender needs n buffers to hold the unacknowledged frames. If the window ever grows to its maximum size, the sending data link layer must forcibly shut off the network layer until another buffer becomes free.
- The receiving data link layer's window corresponds to the frames it may accept. Any frame falling within the window is put in the receiver's buffer. When a frame whose sequence number is equal to the lower edge of the window is re- ceived, it is passed to the network layer and the window is rotated by one. Any frame falling outside the window is discarded. In all of these cases, a subsequent acknowledgement is generated so that the sender may work out how to proceed.

```
/* Protocol 4 (Sliding window) is bidirectional. */

#define MAX_SEQ 1                                   /* must be 1 for protocol 4 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"
void protocol4 (void)
{
  seq_nr next_frame_to_send;                        /* 0 or 1 only */
  seq_nr frame_expected;                            /* 0 or 1 only */
  frame r, s;                                       /* scratch variables */
  packet buffer;                                    /* current packet being sent */
  event_type event;
  next_frame_to_send = 0;                           /* next frame on the outbound stream */
  frame_expected = 0;                               /* frame expected next */
  from_network_layer(&buffer);                      /* fetch a packet from the network layer */
  s.info = buffer;                                  /* prepare to send the initial frame */
  s.seq = next_frame_to_send;                       /* insert sequence number into frame */
  s.ack = 1 - frame_expected;                       /* piggybacked ack */
  to_physical_layer(&s);                            /* transmit the frame */
  start_timer(s.seq);                               /* start the timer running */
  while (true) {
      wait_for_event(&event);                       /* frame_arrival, cksum_err, or timeout */
      if (event == frame_arrival) {                 /* a frame has arrived undamaged */
          from_physical_layer(&r);                  /* go get it */
          if (r.seq == frame_expected) {            /* handle inbound frame stream */
              to_network_layer(&r.info);            /* pass packet to network layer */
              inc(frame_expected);                  /* invert seq number expected next */
          }
          if (r.ack == next_frame_to_send) {        /* handle outbound frame stream */
              stop_timer(r.ack);                    /* turn the timer off */
              from_network_layer(&buffer);          /* fetch new pkt from network layer */
              inc(next_frame_to_send);              /* invert sender's sequence number */
          }
      }
      s.info = buffer;                              /* construct outbound frame */
      s.seq = next_frame_to_send;                   /* insert sequence number into it */
      s.ack = 1 - frame_expected;                   /* seq number of last received frame */
      to_physical_layer(&s);                        /* transmit a frame */
      start_timer(s.seq);                           /* start the timer running */
  }
}
```

1 bit sliding window protocol

Under normal circumstances, one of the two data link layers goes first and transmits the first frame. In other words, only one of the data link layer programs should contain the to physical layer and start timer procedure calls outside the main loop. The starting machine fetches the first packet from its network layer, builds a frame from it, and sends it. When this (or any) frame arrives, the receiv- ing data link layer checks to see if it is a duplicate, just as in protocol 3. If the frame is the one expected, it is passed to the network layer and the receiver's win- dow is slid up.

- consider a 50-kbps satellite channel with a 500-msec round-trip propaga-

tion delay. Let us imagine trying to use protocol 4 to send 1000-bit frames via the satellite. At t = 0 the sender starts sending the first frame. At t = 20 msec the frame has been com- pletely sent. Not until t = 270 msec has the frame fully arrived at the receiver, and not until t = 520 msec has the acknowledgement arrived back at the sender, under the best of circumstances (of no waiting in the receiver and a short ac- knowledgement frame). This means that the sender was blocked 500/520 or 96% of the time. In other words, only 4% of the available bandwidth was used. Clearly, the combination of a long transit time, high bandwidth, and short frame length is disastrous in terms of efficiency. Basically, the solution lies in allowing the sender to transmit up to w frames before blocking, instead of just 1.

To find an appropriate value for w we need to know how many frames can fit inside the channel as they propagate from sender to receiver. This capacity is de- termined by the bandwidth in bits/sec multiplied by the one-way transit time, or the bandwidth-delay product of the link. We can divide this quantity by the number of bits in a frame to express it as a number of frames. Call this quantity BD. Then w should be set to 2BD + 1. Twice the bandwidth-delay is the number of frames that can be outstanding if the sender continuously sends frames when the round-trip time to receive an acknowledgement is considered. The "+1" is because an acknowledgement frame will not be sent until after a complete frame is received. For the example link with a bandwidth of 50 kbps and a one-way transit time of 250 msec, the bandwidth-delay product is 12.5 kbit or 12.5 frames of 1000 bits each. 2BD + 1 is then 26 frames. Assume the sender begins sending frame 0 as before and sends a new frame every 20 msec. By the time it has finished sending 26 frames, at t = 520 msec, the acknowledgement for frame 0 will have just arri- ved. Thereafter, acknowledgements will arrive every 20 msec, so the sender will always get permission to continue just when it needs it. From then onwards, 25 or 26 unacknowledged frames will always be outstanding. Put in other terms, the sender's maximum window size is 26. For smaller window sizes, the utilization of the link will be less than 100% since the sender will be blocked sometimes. We can write the utilization as the fraction of time that the sender is not blocked:

link utilization $\leq \frac{w}{1+2BD}$

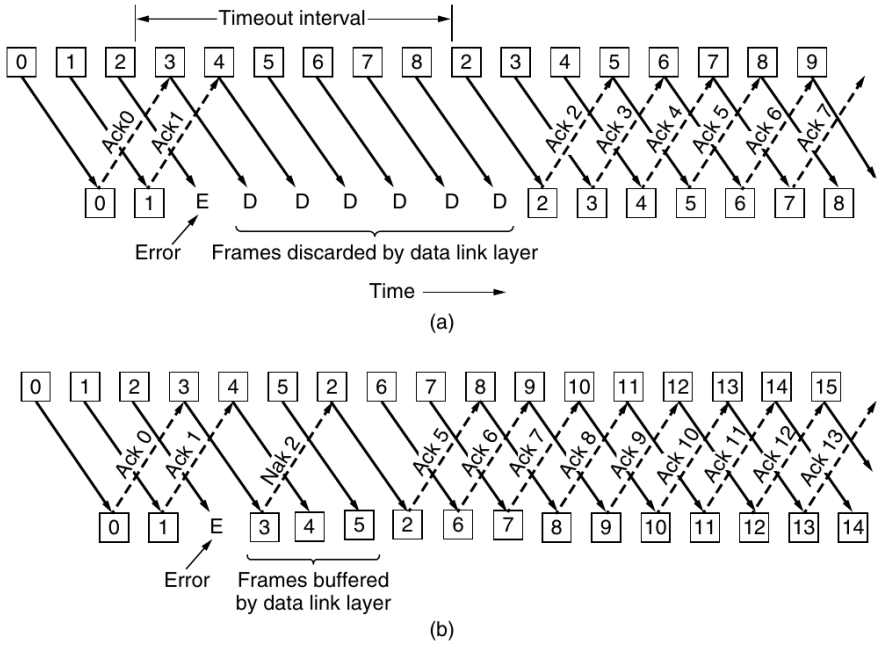
This value is an upper bound because it does not allow for any frame processing time and treats the acknowledgement frame as having zero length, since it is usually short.

This technique of keeping multiple frames in flight is an example of pipelining.

Pipelining frames over an unreliable communication channel raises some serious issues. First, what happens if a frame in the middle of a long stream is damaged or lost? Large numbers of succeeding frames will arrive at the receiver before the sender even finds out that anything is wrong. When a damaged frame arrives at the receiver, it obviously should be discarded, but what should the re- ceiver

do with all the correct frames following it? Remember that the receiving data link layer is obligated to hand packets to the network layer in sequence.

Two basic approaches are available for dealing with errors in the presence of pipelining, both of which are shown in below fig.



Pipelining and error recovery. Effect of an error when (a) receiver's window size is 1 and (b) receiver's window size is large.

One option, called go-back-n, is for the receiver simply to discard all subsequent frames, sending no acknowledgements for the discarded frames. This strategy corresponds to a receive window of size 1. In other words, the data link layer refuses to accept any frame except the next one it must give to the network layer. If the sender's window fills up before the timer runs out, the pipeline will begin to empty. Eventually, the sender will time out and retransmit all unacknowledged frames in order, starting with the damaged or lost one. This approach can waste a lot of bandwidth if the error rate is high.

In Fig. 3-18(b) we see go-back-n for the case in which the receiver's window is large. Frames 0 and 1 are correctly received and acknowledged. Frame 2, however, is damaged or lost. The sender, unaware of this problem, continues to send frames until the timer for frame 2 expires. Then it backs up to frame 2 and starts over with it, sending 2, 3, 4, etc. all over again. The other general strategy for handling errors when frames are pipelined is called selective repeat. When it is used, a bad frame that is received is discarded, but any good frames received after it are accepted and buffered. When the sender times out, only the

oldest unacknowledged frame is retransmitted. If that frame arrives correctly, the receiver can deliver to the network layer, in sequence, all the frames it has buffered. Selective repeat corresponds to a receiver window larger than 1. This approach can require large amounts of data link layer memory if the window is large. Selective repeat is often combined with having the receiver send a negative acknowledgement (NAK) when it detects an error, for example, when it receives a checksum error or a frame out of sequence. NAKs stimulate retransmission be- fore the corresponding timer expires and thus improve performance. In Fig. 3-18(b), frames 0 and 1 are again correctly received and acknowledged and frame 2 is lost. When frame 3 arrives at the receiver, the data link layer there notices that it has missed a frame, so it sends back a NAK for 2 but buffers 3. When frames 4 and 5 arrive, they, too, are buffered by the data link layer instead of being passed to the network layer. Eventually, the NAK 2 gets back to the sender, which immediately resends frame 2. When that arrives, the data link layer now has 2, 3, 4, and 5 and can pass all of them to the network layer in the correct order. It can also acknowledge all frames up to and including 5, as shown in the figure. If the NAK should get lost, eventually the sender will time out for frame 2 and send it (and only it) of its own accord, but that may be a quite a while later.

Network Layer

Network links can be divided into two categories: those using point-to-point connections and those using broadcast channels. We studied point-to-point links in Physical Layer chapter; this chapter deals with broadcast links and their protocols.

In the literature, broadcast channels are sometimes referred to as multiaccess channels or random access channels.

The protocols used to determine who goes next on a multiaccess channel belong to a sublayer of the data link layer called the MAC (Medium Access Control) sublayer.

Static Channel Allocation

The traditional way of allocating a single channel, such as a telephone trunk, among multiple competing users is to chop up its capacity by using one of the multiplexing schemes we described previously, such as FDM (Frequency Division Multiplexing). If there are N users, the bandwidth is divided into N equal-sized portions, with each user being assigned one portion. Since each user has a private frequency band, there is now no interference among users. When there is only a small and constant number of users, each of which has a steady stream or a heavy load of traffic, this division is a simple and efficient allocation mechanism. A wireless example is FM radio stations. Each station gets a portion of the FM band and uses it most of the time to broadcast its signal.

However,when the number of senders is large and varying or the traffic is bursty, FDM presents some problems. If the spectrum is cut up into N regions and fewer than N users are currently interested in communicating, a large piece of valuable spectrum will be wasted. And if more than N users want to communicate, some of them will be denied permission for lack of bandwidth, even if some of the users who have been assigned a frequency band hardly ever transmit or receive anything.

The poor performance of static FDM can easily be seen with a simple queueing theory calculation. Let us start by finding the mean time delay, T, to send a frame onto a channel of capacity C bps. We assume that the frames arrive

randomly with an average arrival rate of frames/sec, and that the frames vary in length with an average length of 1/ bits.

With these parameters, the arrival rate of the channel is / bps. Then, the system utilization is given by = arrival rate/service rate = / C . A standard queueing theory result is

$T = \frac{1/service\_rate}{1-\rho}$

Now let us divide the single channel into N independent subchannels, each with capacity C /N bps. The mean input rate on each of the subchannels will now be /N. Recomputing T, we get

$T_N = NT.$

The mean delay for the divided channel is N times worse than if all the frames were somehow magically arranged orderly in a big central queue. This same result says that a bank lobby full of ATM machines is better off having a single queue feeding all the machines than a separate queue in front of each machine.

Precisely the same arguments that apply to FDM also apply to other ways of statically dividing the channel. If we were to use time division multiplexing (TDM) and allocate each user every Nth time slot, if a user does not use the allocated slot, it would just lie fallow.

Since none of the traditional static channel allocation methods work well at all with bursty traffic, we will now explore dynamic methods.

Assumptions for Dynamic Channel Allocation

Independent Traffic. The model consists of N independent stations (e.g., computers, telephones), each with a program or user that generates frames for transmission. The expected number of frames generated in an interval of length Δt is Δt, where is a constant (the arrival rate of new frames). Once a frame has been generated, the station is blocked and does nothing until the frame has been successfully transmitted.

Single Channel. A single channel is available for all communication. All stations can transmit on it and all can receive from it. The stations are assumed to be equally capable, though protocols may assign them different roles (e.g., priorities).

Observable Collisions. If two frames are transmitted simultaneously, they overlap in time and the resulting signal is garbled. This event is called a collision. All stations can detect that a collision has occurred. A collided frame must be transmitted again later. No errors other than those generated by collisions occur. (collisions are usually inferred after the fact by the lack of an expected acknowledgement frame)

Continuous or Slotted Time. Time may be assumed continuous, in which case frame transmission can begin at any instant. Alternatively, time may be slotted or divided into discrete intervals (called slots). Frame transmissions must then begin at the start of a slot. A slot may contain 0, 1, or more frames, corresponding to an idle slot, a successful transmission, or a collision, respectively.

Carrier Sense or No Carrier Sense. With the carrier sense assumption, stations can tell if the channel is in use before trying to use it. No station will attempt to use the channel while it is sensed as busy. If there is no carrier sense, stations cannot sense the channel before trying to use it. They just go ahead and transmit. Only later can they determine whether the transmission was successful.

Pure Aloha

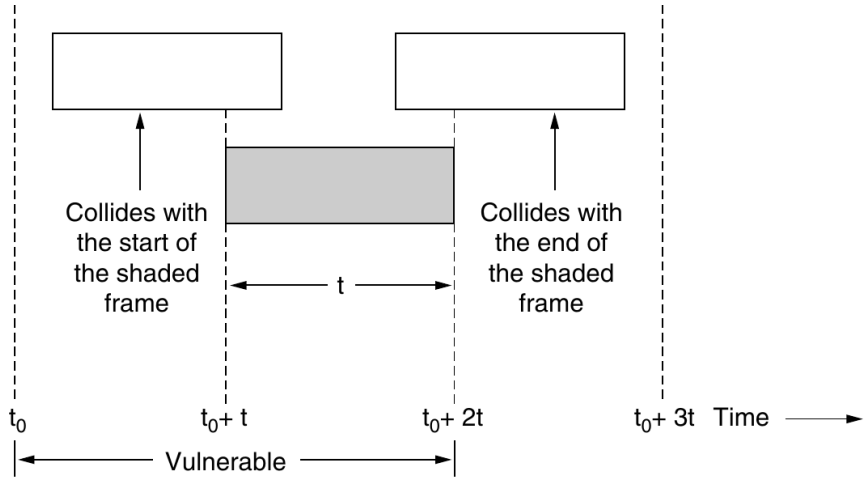
The basic idea of an ALOHA system is simple: let users transmit whenever they have data to be sent. There will be collisions, of course, and the colliding frames will be damaged. Senders need some way to find out if this is the case. In the ALOHA system, after each station has sent its frame to the central computer, this computer rebroadcasts the frame to all of the stations. A sending station can thus listen for the broadcast from the hub to see if its frame has gotten through. In other systems, such as wired LANs, the sender might be able to listen for collisions while transmitting. If the frame was destroyed, the sender just waits a random amount of time and sends it again. The waiting time must be random or the same frames will collide over and over, in lockstep. Systems in which multiple users share a common channel in a way that can lead to conflicts are known as contention systems.

Analysis:

Let the "frame time" denote the amount of time needed to transmit the standard, fixed-length frame (i.e., the frame length divided by the bit rate). At this point, we assume that the new frames generated by the stations are well modeled by a Poisson distribution with a mean of N frames per frame time. (The infinitepopulation assumption is needed to ensure that N does not decrease as users become blocked.) If $N > 1$, the user community is generating frames at a higher rate than the channel can handle, and nearly every frame will suffer a collision. For reasonable throughput, we would expect $0 < N < 1$.

In addition to the new frames, the stations also generate retransmissions of frames that previously suffered collisions. Let us further assume that the old and new frames combined are well modeled by a Poisson distribution, with mean of G frames per frame time. Clearly, G N. At low load (i.e., N 0), there will be few collisions, hence few retransmissions, so G N. At high load, there will be many collisions, so G > N. Under all loads, the throughput, S, is just the offered load, G, times the probability, P_0 , of a transmission succeeding—that is, $S = GP_0$, where P_0 is the probability that a frame does not suffer a collision.

Let t be the time required to send one frame



The probability that k frames are generated during a given frame time, in which G frames are expected, is given by the Poisson distribution

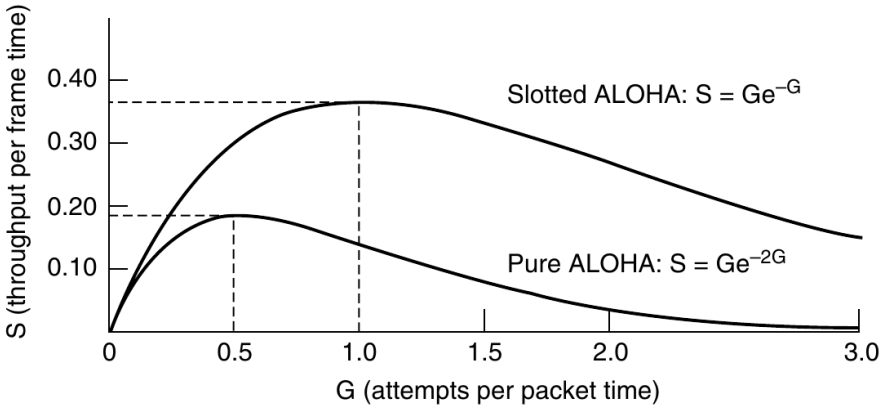
$Pr[k] = \frac{G^k e^{-G}}{k!}$

so the probability of zero frames is just e^{-G}. In an interval two frame times long, the mean number of frames generated is $2G$. The probability of no frames being initiated during the entire vulnerable period is thus given by $P_0 = e^{-2G}$. Using $S = GP_0$, we get $S = Ge^{-2G}$.

Slotted Aloha

One way to achieve synchronization would be to have one special station emit a pip at the start of each interval, like a clock.

The probability of no other traffic during the same slot as our test frame is then e^{-G} , which leads to $S = Ge^{-G}$



Operating at higher values of G reduces the number of empties but increases the number of collisions exponentially. To see how this rapid growth of collisions with G comes about, consider the transmission of a test frame. The probability that it will avoid a collision is e^{-G} , which is the probability that all the other stations are silent in that slot. The probability of a collision is then just $1-e^{-G}$. The probability of a transmission requiring exactly k attempts (i.e., k − 1 collisions followed by one success) is

$P_k = e^{-G}(1 - e^{-G})^{k-1}$

The expected number of transmissions, E, per line typed at a terminal is then

$E = \sum_{k=1}^{\inf} kP_k = \sum_{k=1}^{\inf} ke^{-G}(1 - e^{-G})^{k-1} = e^G.$

As a result of the exponential dependence of E upon G, small increases in the channel load can drastically reduce its performance.

1 persistent CSMA

- CSMA = Carrier Sense Multiple Access.
- When a station has data to send, it first listens to the chan- nel to see if anyone else is transmitting at that moment. If the channel is idle, the stations sends its data. Otherwise, if the channel is busy, the station just waits until it becomes idle. Then the station transmits a frame. If a collision occurs, the station waits a random amount of time and starts all over again. The protocol is called 1-persistent because the station transmits with a probability of 1 when it finds the channel idle.

More subtly, the propagation delay has an important effect on collisions. There is a chance that just after a station begins sending, another station will become ready to send and sense the channel. If the first station's signal has not yet reached the second one, the latter will sense an idle channel and will also

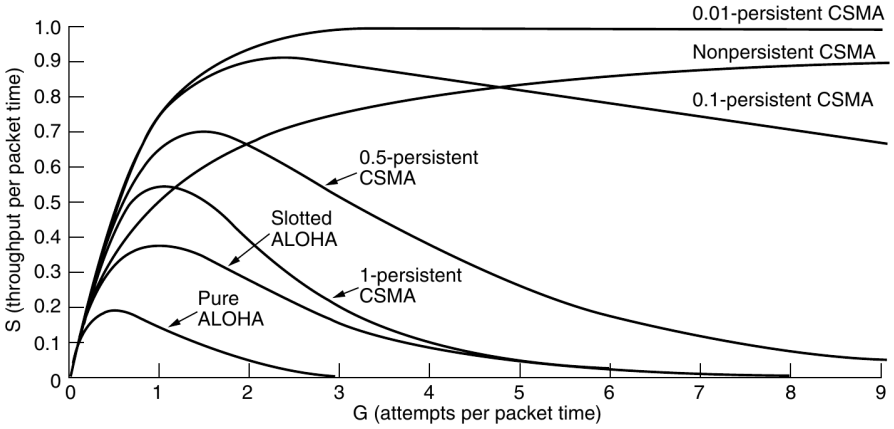
begin sending, resulting in a collision. This chance depends on the number of frames that fit on the channel, or the bandwidth-delay product of the channel. If only a tiny fraction of a frame fits on the channel, which is the case in most LANs since the propagation delay is small, the chance of a collision happening is small. The larger the bandwidth-delay product, the more important this effect becomes, and the worse the performance of the protocol.

Non persistent CSMA

In this protocol, a conscious attempt is made to be less greedy than in the previous one. As before, a station senses the channel when it wants to send a frame, and if no one else is sending, the station begins doing so itself. However, if the channel is already in use, the station does not continually sense it for the purpose of seizing it im- mediately upon detecting the end of the previous transmission. Instead, it waits a random period of time and then repeats the algorithm. Consequently, this algo- rithm leads to better channel utilization but longer delays than 1-persistent CSMA.

p persistent CSMA

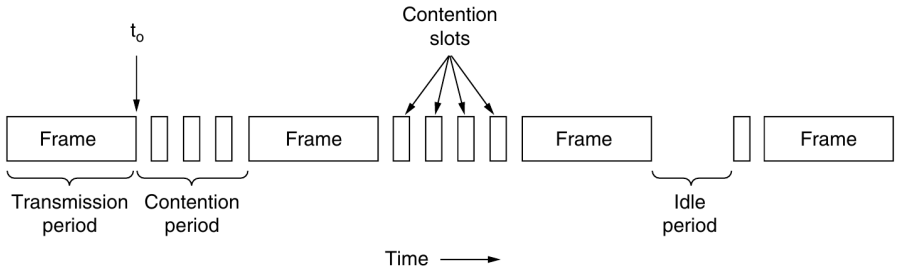
It applies to slotted channels and works as follows. When a station becomes ready to send, it senses the channel. If it is idle, it transmits with a probability p. With a probability q = 1 − p, it defers until the next slot. If that slot is also idle, it either transmits or defers again, with probabilities p and q. This process is repeated until either the frame has been transmitted or another station has begun transmitting. In the latter case, the unlucky station acts as if there had been a collision (i.e., it waits a random time and starts again). If the station initially senses that the channel is busy, it waits until the next slot and applies the above algorithm.



CSMA with Collision Detection (CSMA/CD)

Another improvement is for the stations to quickly detect the collision and abruptly stop transmitting, (rather than finishing them) since they are irretrievably garbled anyway. This strategy saves time and bandwidth.

It is important to realize that collision detection is an analog proc- ess. The station's hardware must listen to the channel while it is transmitting. If the signal it reads back is different from the signal it is putting out, it knows that a collision is occurring. The implications are that a received signal must not be tiny compared to the transmitted signal (which is difficult for wireless, as received sig- nals may be 1,000,000 times weaker than transmitted signals) and that the modu- lation must be chosen to allow collisions to be detected (e.g., a collision of two 0- volt signals may well be impossible to detect).



At the point marked t_0 , a station has finished transmitting its frame. Any other station having a frame to send may now attempt to do so. If two or more stations decide to transmit simultaneously, there will be a collision. If a sta- tion detects a collision, it aborts its transmission, waits a random period of time, and then tries again (assuming that no other station has started transmitting in the meantime). Therefore, our model for CSMA/CD will consist of alternating contention and transmission periods, with idle periods occurring when all stations are quiet (e.g., for lack of work).

Now let us look at the details of the contention algorithm. Suppose that two stations both begin transmitting at exactly time t_0 . How long will it take them to realize that they have collided? The answer is vital to determining the length of the contention period and hence what the delay and throughput will be. The minimum time to detect the collision is just the time it takes the signal to propagate from one station to the other. Based on this information, you might think that a station that has not heard a collision for a time equal to the full cable propagation time after starting its transmission can be sure it has seized the cable. By "seized," we mean that all other stations know it is transmitting and will not interfere. This conclusion is wrong. Consider the following worst-case scenario. Let the time for a signal to pro- pagate between the two farthest stations be . At t_0 , one station begins trans- mitting. At $t_0 +$ $-$, an instant before the signal arrives at the most distant sta- tion, that station also begins transmitting. Of course, it detects the collision al- most

instantly and stops, but the little noise burst caused by the collision does not get back to the original station until time 2 − . In other words, in the worst case a station cannot be sure that it has seized the channel until it has transmitted for 2 without hearing a collision.

Collision free protocols

Although collisions do not occur with CSMA/CD once a station has unambiguously captured the channel, they can still occur during the contention period. These collisions adversely affect the system performance, especially when the bandwidth-delay product is large, such as when the cable is long (i.e., large) and the frames are short.

In the protocols to be described, we assume that there are exactly N stations, each programmed with a unique address from 0 to N − 1. It does not matter that some stations may be inactive part of the time. We also assume that propagation delay is negligible. We continue using the model of previous figure with its discrete contention slots.

Bit Map Protocol

In our first collision-free protocol, the basic bit-map method, each con- tention period consists of exactly N slots. If station 0 has a frame to send, it trans- mits a 1 bit during the slot 0. No other station is allowed to transmit during this slot. Regardless of what station 0 does, station 1 gets the opportunity to transmit a 1 bit during slot 1, but only if it has a frame queued. In general, station j may announce that it has a frame to send by inserting a 1 bit into slot j. After all N slots have passed by, each station has complete knowledge of which stations wish to transmit. At that point, they begin transmitting frames in numerical order.

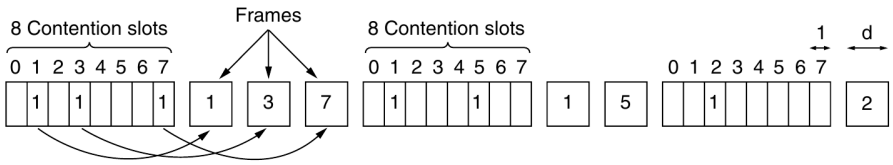


Figure 1: basic bit map protocol

Protocols like this in which the desire to transmit is broadcast before the actual transmission are called reservation protocols because they reserve channel ownership in advance and prevent collisions.

Analysis:

Under conditions of low load, the bit map will simply be repeated over and over, for lack of data frames. Consider the situation from the point of view of a low-numbered station, such as 0 or 1. Typically, when it becomes ready to send, the "current" slot will be somewhere in the middle of the bit map. On average, the station will have to wait N /2 slots for the current scan to finish and another full N slots for the following scan to run to completion before it may begin trans- mitting. The prospects for high-numbered stations are brighter. Generally, these will only have to wait half a scan (N /2 bit slots) before starting to transmit. High- numbered stations rarely have to wait for the next scan. Since low-numbered sta- tions must wait on average 1.5N slots and high-numbered stations must wait on average 0.5N slots, the mean for all stations is N slots. The channel efficiency at low load is easy to compute. The overhead per frame is N bits and the amount of data is d bits, for an efficiency of d /(d + N). At high load, when all the stations have something to send all the time, the N- bit contention period is prorated over N frames, yielding an overhead of only 1 bit per frame, or an efficiency of d /(d + 1). The mean delay for a frame is equal to the sum of the time it queues inside its station, plus an additional $(N - 1)d + N$ once it gets to the head of its internal queue. This interval is how long it takes to wait for all other stations to have their turn sending a frame and another bitmap.

Token Passing

The token represents permission to send. If a station has a frame queued for transmission when it receives the token, it can send that frame before it passes the token to the next station. If it has no queued frame, it simply passes the token.

In a token ring protocol, the topology of the network is used to define the order in which stations send. Frames are also transmitted in the direction of the token. This way they will circulate around the ring and reach whichever station is the destination. How- ever, to stop the frame circulating indefinitely (like the token), some station needs to remove it from the ring. This station may be either the one that originally sent the frame, after it has gone through a complete cycle, or the station that was the intended recipient of the frame.

Note that we do not need a physical ring to implement token passing. The channel connecting the stations might instead be a single long bus. Each station then uses the bus to send the token to the next station in the predefined sequence. Possession of the token allows a station to use the bus to send one frame, as be- fore. This protocol is called token bus.

The performance of token passing is similar to that of the bit-map protocol, though the contention slots and frames of one cycle are now intermingled. After sending a frame, each station must wait for all N stations (including itself) to send the token to their neighbors and the other $N - 1$ stations to send a frame,

if they have one. A subtle difference is that, since all positions in the cycle are equiva- lent, there is no bias for low- or high-numbered stations

Binary Countdown

A problem with the basic bit-map protocol, and by extension token passing, is that the overhead is 1 bit per station, so it does not scale well to networks with thousands of stations. We can do better than that by using binary station addresses with a channel that combines transmissions. A station wanting to use the channel now broadcasts its address as a binary bit string, starting with the high- order bit. All addresses are assumed to be the same length. The bits in each ad- dress position from different stations are BOOLEAN ORed together by the chan- nel when they are sent at the same time.

Arbitration: the use of an arbitrator (an independent person or body officially appointed to settle a dispute.) to settle a dispute.

To avoid conflicts, an arbitration rule must be applied: as soon as a station sees that a high-order bit position that is 0 in its address has been overwritten with a 1, it gives up. For example, if stations 0010, 0100, 1001, and 1010 are all trying to get the channel, in the first bit time the stations transmit 0, 0, 1, and 1, re- spectively. These are ORed together to form a 1. Stations 0010 and 0100 see the 1 and know that a higher-numbered station is competing for the channel, so they give up for the current round. Stations 1001 and 1010 continue. The next bit is 0, and both stations continue. The next bit is 1, so station 1001 gives up. The winner is station 1010 because it has the highest address. After winning the bidding, it may now transmit a frame, after which another bidding cycle starts.

The channel efficiency of this method is d /(d + log2 N). If, however, the frame format has been cleverly chosen so that the sender's address is the first field in the frame, even these log2 N bits are not wasted, and the efficiency is 100%.

Wireless LAN protocols

Wireless LAN — a set of nodes sending messages to each other via a wireless medium.

A station on a wireless LAN may not be able to transmit frames to or receive frames from all other stations because of the limited radio range of the stations..

Detection of collision is difficult and often impossible. Acknowledgments are used to discover collisions and other errors.

We will assume that each radio transmitter has some fixed range, represented by a circular coverage region within which another station can sense and receive the station's transmission.

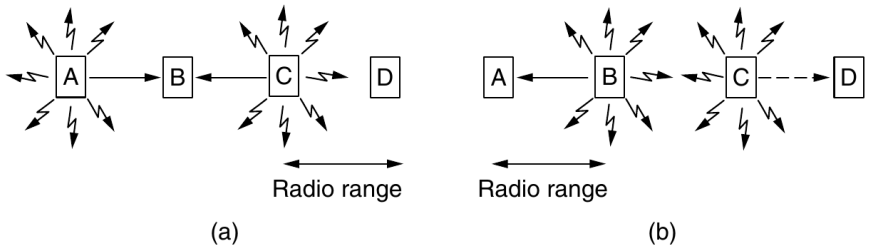


Figure 4-11. A wireless LAN. (a) *A* and *C* are hidden terminals when transmitting to *B*. (b) *B* and *C* are exposed terminals when transmitting to *A* and *D*.

First consider what happens when A and C transmit to B, as depicted in Fig. 4-11(a). If A sends and then C immediately senses the medium, it will not hear A because A is out of range. Thus C will falsely conclude that it can transmit to B. If C does start transmitting, it will interfere at B, wiping out the frame from A.

The problem of a station not being able to detect a potential competitor for the medium because the competitor is too far away is called the hidden terminal problem.

Now let us look at a different situation: B transmitting to A at the same time that C wants to transmit to D, as shown in Fig. 4-11(b). If C senses the medium, it will hear a transmission and falsely conclude that it may not send to D (shown as a dashed line). In fact, such a transmission would cause bad reception only in the zone between B and C, where neither of the intended receivers is located. We want a MAC protocol that prevents this kind of deferral from happening because it wastes bandwidth. The problem is called the exposed terminal problem.

MACA

An early and influential protocol that tackles these problems for wireless LANs is MACA (Multiple Access with Collision Avoidance) (Karn, 1990). The basic idea behind it is for the sender to stimulate the receiver into outputting a short frame, so stations nearby can detect this transmission and avoid transmitting for the duration of the upcoming (large) data frame. This technique is used instead of carrier sense. MACA is illustrated in Fig. 4-12. Let us see how A sends a frame to B. A starts by sending an RTS (Request To Send) frame to B, as shown in Fig. 4-12(a). This short frame (30 bytes) contains the length of the data frame that will eventu- ally follow. Then B replies with a CTS (Clear To Send) frame, as shown in Fig. 4-12(b). The CTS frame contains the data length (copied from the RTS frame). Upon receipt of the CTS frame, A begins transmission.

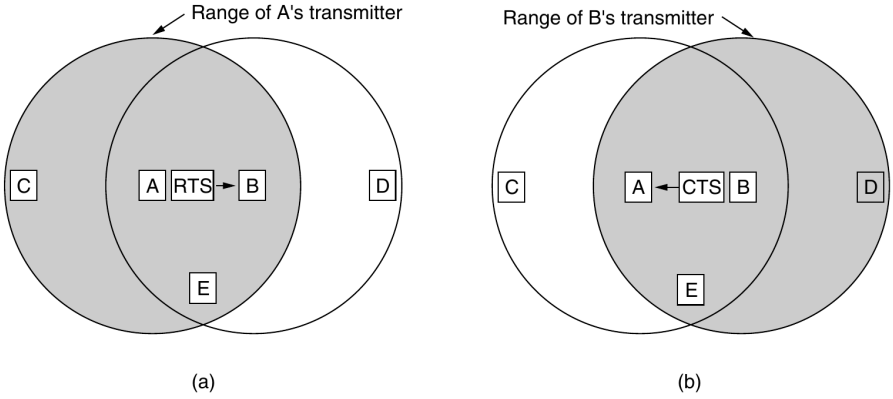


Figure 4-12. The MACA protocol. (a) *A* sending an RTS to *B*. (b) *B* responding with a CTS to *A*.

Now let us see how stations overhearing either of these frames react. Any station hearing the RTS is clearly close to A and must remain silent long enough for the CTS to be transmitted back to A without conflict. Any station hearing the CTS is clearly close to B and must remain silent during the upcoming data trans- mission, whose length it can tell by examining the CTS frame. In Fig. 4-12, C is within range of A but not within range of B. Therefore, it hears the RTS from A but not the CTS from B. As long as it does not interfere with the CTS, it is free to transmit while the data frame is being sent. In contrast, D is within range of B but not A. It does not hear the RTS but does hear the CTS. Hearing the CTS tips it off that it is close to a station that is about to receive a frame, so it defers sending anything until that frame is expected to be finished. Station E hears both control messages and, like D, must be silent until the data frame is complete.

802.11 MAC Sublayer Protocol

A station that has a frame to send starts with a ran- dom backoff (except in the case that it has not used the channel recently and the channel is idle). It does not wait for a collision. The number of slots to backoff is chosen in the range 0 to, say, 15 in the case of the OFDM physical layer. The sta- tion waits until the channel is idle, by sensing that there is no signal for a short period of time (called the DIFS, as we explain below), and counts down idle slots, pausing when frames are sent. It sends its frame when the counter reaches 0. If the frame gets through, the destination immediately sends a short acknowledge- ment. Lack of an acknowledgement is inferred to indicate an error, whether a col- lision or otherwise. In this case, the sender doubles the backoff period and tries again, continuing with exponential backoff as in Ethernet until the frame has been successfully transmitted or the maximum number of retransmissions has been reached.

This mode of operation is called DCF (Distributed Coordination Function) because each station acts independently, without any kind of central control. The standard also includes an optional mode of operation called PCF (Point Coordination Function) in which the access point controls all activity in its cell, just like a cellular base station.

To reduce ambiguities about which station is sending, 802.11 defines channel sensing to consist of both physical sensing and virtual sensing. Physical sensing simply checks the medium to see if there is a valid signal. With virtual sensing, each station keeps a logical record of when the channel is in use by tracking the NAV (Network Allocation Vector). Each frame carries a NAV field that says how long the sequence of which this frame is part will take to complete. Stations that overhear this frame know that the channel will be busy for the period indi- cated by the NAV , regardless of whether they can sense a physical signal. For ex- ample, the NAV of a data frame includes the time needed to send an acknowledge- ment. All stations that hear the data frame will defer during the acknowledgement period, whether or not they can hear the acknowledgement. An optional RTS/CTS mechanism uses the NAV to prevent terminals from sending frames at the same time as hidden terminals. It is shown in Fig. 4-27. In this example, A wants to send to B. C is a station within range of A (and possibly within range of B, but that does not matter). D is a station within range of B but not within range of A. The protocol starts when A decides it wants to send data to B. A begins by sending an RTS frame to B to request permission to send it a frame. If B receives this request, it answers with a CTS frame to indicate that the channel is clear to send. Upon receipt of the CTS , A sends its frame and starts an ACK timer. Upon correct receipt of the data frame, B responds with an ACK frame, completing the exchange. If A's ACK timer expires before the ACK gets back to it, it is treated as a collision and the whole protocol is run again after a backoff.

Now let us consider this exchange from the viewpoints of C and D. C is with- in range of A, so it may receive the RTS frame. If it does, it realizes that someone is going to send data soon. From the information provided in the RTS request, it can estimate how long the sequence will take, including the final ACK. So, for the good of all, it desists from transmitting anything until the exchange is completed. It does so by updating its record of the NAV to indicate that the channel is busy, as shown in Fig. 4-27. D does not hear the RTS, but it does hear the CTS, so it also updates its NAV. Note that the NAV signals are not transmitted; they are just in- ternal reminders to keep quiet for a certain period of time.

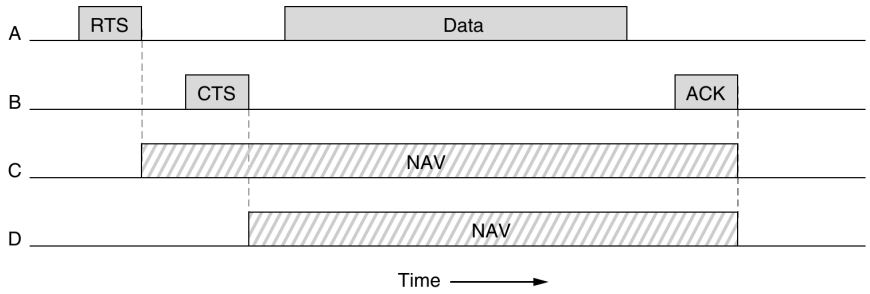


Figure 4-27. Virtual channel sensing using CSMA/CA.

RTS/CTS in 802.11 is a little different than in the MACA protocol we saw in Sec 4.2 because everyone hearing the RTS or CTS remains quiet for the duration to allow the ACK to get through without collision. Because of this, it does not help with exposed terminals as MACA did, only with hidden terminals.

After a frame has been sent, a certain amount of idle time is required before any station may send a frame to check that the channel is no longer in use. The trick is to define different time intervals for different kinds of frames.

Five intervals are depicted in Fig. 4-28. The interval between regular data frames is called the DIFS (DCF InterFrame Spacing). Any station may attempt to acquire the channel to send a new frame after the medium has been idle for DIFS. The usual contention rules apply, and binary exponential backoff may be needed if a collision occurs. The shortest interval is SIFS (Short InterFrame Spacing). It is used to allow the parties in a single dialog the chance to go first. Examples include letting the receiver send an ACK , other control frame sequences like RTS and CTS, or letting a sender transmit a burst of fragments. Sending the next fragment after waiting only SIFS is what prevents another station from jump- ing in with a frame in the middle of the exchange.

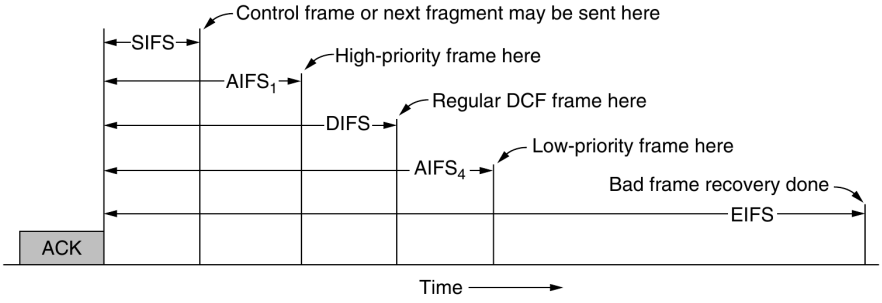


Figure 4-28. Interframe spacing in 802.11.

The two AIFS (Arbitration InterFrame Space) intervals show examples of two different priority levels. The short interval, AIFS1 , is smaller than DIFS but longer than SIFS. It can be used by the AP to move voice or other high-priority traffic to the head of the line. The AP will wait for a shorter interval before it sends the voice traffic, and thus send it before regular traffic. The long interval, AIFS 4 , is larger than DIFS. It is used for background traffic that can be deferred until after regular traffic. The AP will wait for a longer interval before it sends this traffic, giving regular traffic the opportunity to transmit first. The complete quality of service mechanism defines four different priority levels that have dif- ferent backoff parameters as well as different idle parameters. The last time interval, EIFS (Extended InterFrame Spacing), is used only by a station that has just received a bad or unknown frame, to report the problem. The idea is that since the receiver may have no idea of what is going on, it should wait a while to avoid interfering with an ongoing dialog between two stations.

Network Layer (Quiz 2 ↓)

See slides from Lec 15 to 24.

See Lec 15.

See Lec 16.

See Lec 17.

See Lec 18.

See Lec 19.

See Lec 20.

See Lec 21.

See Lec 22.

See Lec 23.

See Lec 24.

OSPF - An Interior Gateway Routing Protocol

Inside of its own network (Autonomous Systems ASes = independent networks), an organization can use its own algorithm for internal routing, or **intradomain** routing. An intradomain routing protocol is also called an **interior** gateway protocol (Similarly **interdomain** routing, here all must use the same **exterior gateway protocol** protocol, viz. BGP (Border Gateway Protocol)).

Salient Features

- Dynamic
- Supports routing based on type of service
- Does load balancing (splitting the load over multiple lines)
- Supports hierarchical systems

Algorithm

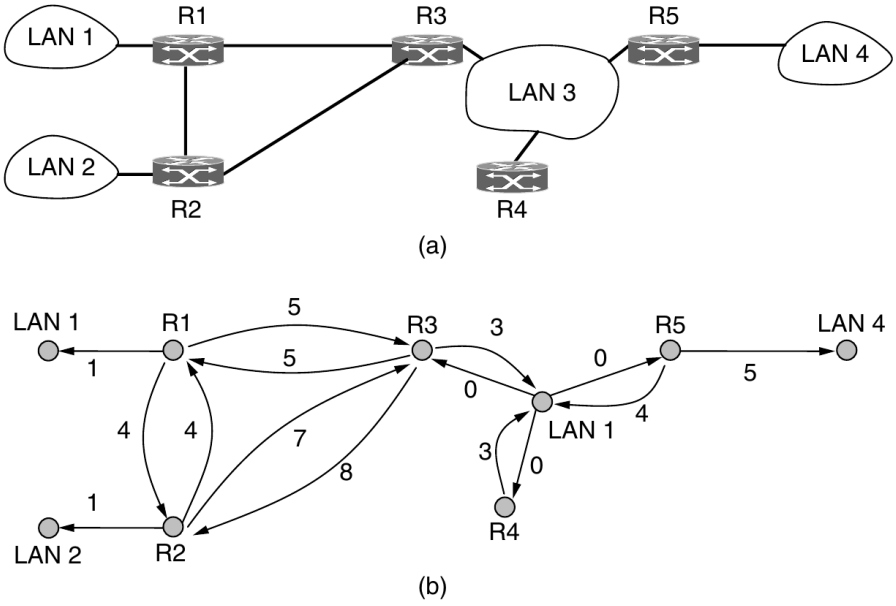


Figure 5-64. (a) An autonomous system. (b) A graph representation of (a).

Figure 2: Fig 5-64

Most of the routers in Fig. 5-64(a) are connected to other routers by point-to-point links, and to networks to reach the hosts on those networks. However, routers R3, R4, and R5 are connected by a broadcast LAN such as switched Ethernet.

A broadcast network is represented by a node for the network itself, plus a node for each router. The arcs from that network node to the routers have weight 0. They are important nonetheless, as without them there is no path through the network. Other networks, which have only hosts, have only an arc reaching them and not one returning. This structure gives routes to hosts, but not through them.

What OSPF fundamentally does is represent the actual network as a graph like this and then use the link state method to have every router compute the shortest path from itself to all other nodes. Multiple paths may be found that are equally short. In this case, OSPF remembers the set of shortest paths and during packet forwarding, traffic is split across them. This helps to balance load. It is called ECMP (Equal Cost MultiPath).

Many of the ASes in the Internet are themselves large and nontrivial to manage. To work at this scale, OSPF allows an AS to be divided into numbered areas, where an area is a network or a set of contiguous networks. Areas do not overlap but need not be exhaustive, that is, some routers may belong to no area. Routers that lie wholly within an area are called **internal** routers. An area is a generalization of an individual network. Outside an area, its destinations are visi- ble but not its topology. This characteristic helps routing to scale.

Every AS has a **backbone** area, called area 0. The routers in this area are called **backbone routers**. All areas are connected to the backbone, possibly by tunnels, so it is possible to go from any area in the AS to any other area in the AS via the backbone. A tunnel is represented in the graph as just another arc with a cost. As with other areas, the topology of the backbone is not visible outside the backbone.

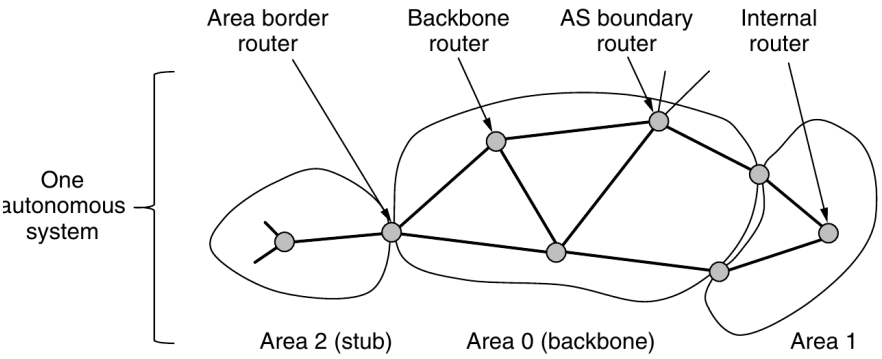


Figure 3: Fig 5-65

Each router that is connected to two or more areas is called an area border router. It must also be part of the backbone. The job of an area border router is to summarize the destinations in one area and to inject this summary into the other areas to which it is connected. This summary includes cost information but not all the details of the topology within an area. Passing cost information allows hosts in other areas to find the best area border router to use to enter an area.

However, if there is only one border router out of an area, even the summary does not need to be passed. Routes to destinations out of the area always start

with the instruction "Go to the border router." This kind of area is called a **stub area**.

The last kind of router is the **AS boundary router**. It injects routes to external destinations on other ASes into the area. The external routes then appear as destinations that can be reached via the AS boundary router with some cost

For a source and destination in the same area, the best intra-area route (that lies wholly within the area) is chosen. For a source and destination in different areas, the inter-area route must go from the source to the backbone, across the backbone to the destination area, and then to the destination.

In particular, it is inefficient to have every router on a LAN talk to every other router on the LAN. To avoid this situation, one router is elected as the designated router. It is said to be adjacent to all the other routers on its LAN, and exchanges information with them. In effect, it is acting as the single node that represents the LAN.

During normal operation, each router periodically floods LINK STATE UPDATE messages to each of its adjacent routers. These messages gives its state and provide the costs used in the topological database. The flooding messages are acknowledged, to make them reliable. Each message has a sequence number, so a router can see whether an incoming LINK STATE UPDATE is older or newer than what it currently has.

DATABASE DESCRIPTION messages give the sequence numbers of all the link state entries currently held by the sender. By comparing its own values with those of the sender, the receiver can determine who has the most recent values.

Either partner can request link state information from the other one by using LINK STATE REQUEST messages. The result of this algorithm is that each pair of adjacent routers checks to see who has the most recent data

Message Type	Description
Hello	Used to discover who the neighbors are
Link state update	Provides the sender's costs to its neighbors
Link state ack	Acknowledges link state update
Database description	Announces which updates the sender has
Link state request	Requests information from the partner

BGP

One common policy is that a customer ISP pays another provider ISP to deliver packets to any other destination on the Internet and receive packets sent from any other destination. The customer ISP is said to buy transit service from the provider ISP.

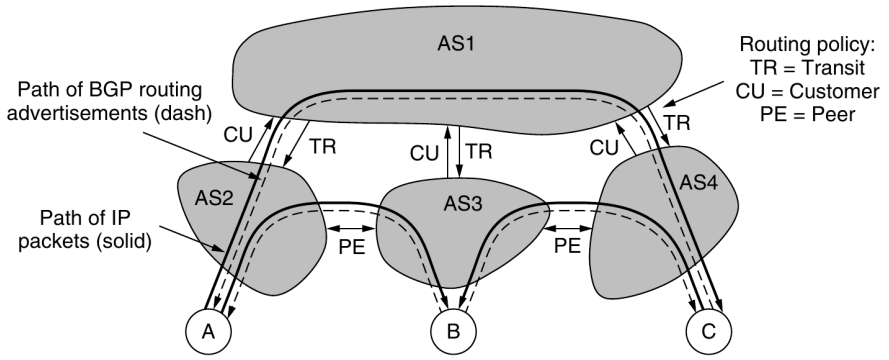


Figure 5-67. Routing policies between four autonomous systems.

Figure 4: BGP

AS2, AS3, and AS4 are customers of AS1. They buy transit service from it. Thus, when source A sends to destination C, the packets travel from AS2 to AS1 and finally to AS4. The routing advertisements travel in the opposite direction to the packets. AS4 advertises C as a destination to its tran- sit provider, AS1, to let sources reach C via AS1. Later, AS1 advertises a route to C to its other customers, including AS2, to let the customers know that they can send traffic to C via AS1.

Suppose that AS2 and AS3 exchange a lot of traffic. Given that their networks are connected already, if they want to, they can use a different policy—they can send traffic directly to each other for free. This will reduce the amount of traffic they must have AS1 deliver on their behalf, and hopefully it will reduce their bills. This policy is called peering. Note that peering is not transitive. AS3 and AS4 also peer with each other. This peering allows traffic from C destined for B to be sent directly to AS4. What hap- pens if C sends a packet to A? AS3 is only advertising a route to B to AS4. It is not advertising a route to A.

On the other hand, some company networks are connected to multiple ISPs. This technique is used to improve reliability, since if the path through one ISP fails, the company can use the path via the other ISP. This technique is called multihoming.

Instead of maintaining just the cost of the route to each destination, each BGP router keeps track of the path used. This approach is called a path vec- tor protocol. The path consists of the next hop router (which may be on the other side of the ISP, not adjacent) and the sequence of ASes, or AS path, that the route has followed (given in reverse order).

So far we have seen how a route advertisement is sent across the link between two ISPs. We still need some way to propagate BGP routes from one side of the ISP to the other, so they can be sent on to the next ISP. This task could be

handled by the intradomain protocol, but because BGP is very good at scaling to large networks, a variant of BGP is often used. It is called iBGP (internal BGP) to distinguish it from the regular use of BGP as eBGP (external BGP)

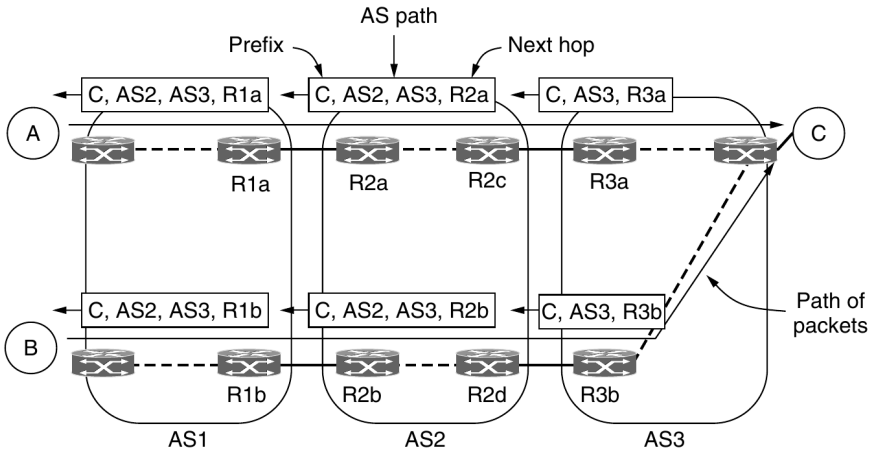


Figure 5-68. Propagation of BGP route advertisements.

Figure 5: BGP2

The final strategy is to prefer the route that has the lowest cost within the ISP. This is the strategy implemented in Fig. 5-68. Packets sent from A to C exit AS1 at the top router, R1a. Packets sent from B exit via the bottom router, R1b. The reason is that both A and B are taking the lowest-cost path or quickest route out of AS1. Because they are located in different parts of the ISP, the quickest exit for each one is different. The same thing happens as the packets pass through AS2. On the last leg, AS3 has to carry the packet from B through its own network. This strategy is known as early exit or hot-potato routing.

Error control and flow control

Given that these mechanisms are used on frames at the link layer, it is natural to wonder why they would be used on segments at the transport layer as well. However, there is little duplication between the link and transport layers in practice. Even though the same mechanisms are used, there are differences in function and degree.

For a difference in function, consider error detection. The link layer check- sum protects a frame while it crosses a single link. The transport layer checksum protects a segment while it crosses an entire network path. It is an end-to-end check, which is not the same as having a check on every link. Saltzer et al. (1984)

describe a situation in which packets were corrupted inside a router. The link layer checksums protected the packets only while they traveled across a link, not while they were inside the router. Thus, packets were delivered incorrectly even though they were correct according to the checks on every link. This and other examples led Saltzer et al. to articulate the end-to-end argument

As a difference in degree, consider retransmissions and the sliding window protocol. Most wireless links, other than satellite links, can have only a single frame outstanding from the sender at a time. That is, the bandwidth-delay product for the link is small enough that not even a whole frame can be stored inside the link. In this case, a small window size is sufficient for good performance. On the other hand, many TCP connections have a bandwidth-delay product that is much larger than a single segment. Consider a connection sending data a- cross the U.S. at 1 Mbps with a round-trip time of 100 msec. Even for this slow connection, 200 Kbit of data will be stored at the receiver in the time it takes to send a segment and receive an acknowledgement. For these situations, a large sliding window must be used. Stop-and-wait will cripple performance.

The best trade-off between source buffering and destination buffering depends on the type of traffic carried by the connection. For low-bandwidth bursty traffic, such as that produced by an interactive terminal, it is reasonable not to dedicate any buffers, but rather to acquire them dynamically at both ends, relying on buff- ering at the sender if segments must occasionally be discarded. On the other hand, for file transfer and other high-bandwidth traffic, it is better if the receiver does dedicate a full window of buffers, to allow the data to flow at maximum speed. This is the strategy that TCP uses.

There still remains the question of how to organize the buffer pool. If most segments are nearly the same size, it is natural to organize the buffers as a pool of identically sized buffers, with one segment per buffer, as in Fig. 6-15(a). Howev- er, if there is wide variation in segment size, from short requests for Web pages to large packets in peer-to-peer file transfers, a pool of fixed-sized buffers presents problems. If the buffer size is chosen to be equal to the largest possible segment, space will be wasted whenever a short segment arrives. If the buffer size is cho- sen to be less than the maximum segment size, multiple buffers will be needed for long segments, with the attendant complexity. Another approach to the buffer size problem is to use variable-sized buffers, as in Fig. 6-15(b). The advantage here is better memory utilization, at the price of more complicated buffer management. A third possibility is to dedicate a single large circular buffer per connection, as in Fig. 6-15(c). This system is simple and elegant and does not depend on segment sizes, but makes good use of memory only when the connections are heavily loaded.

Initially, the sender requests a certain number of buffers, based on its expected needs. The receiver then grants as many of these as it can afford. Every time the sender transmits a segment, it must decrement its allocation, stopping altogether when the allocation reaches zero. The receiver separately piggybacks both acknowledgements and buffer allocations onto the reverse traffic.

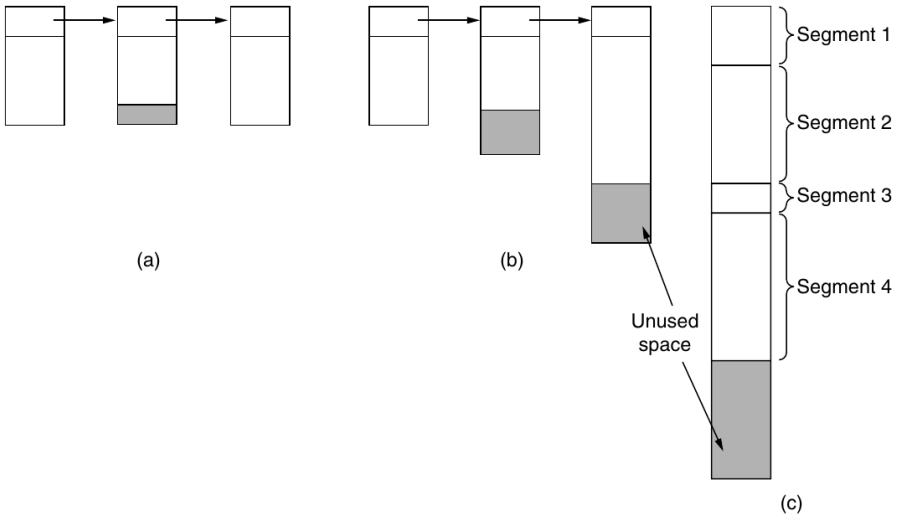


Figure 6-15. (a) Chained fixed-size buffers. (b) Chained variable-sized buffers. (c) One large circular buffer per connection.

Figure 6: Buffer Size illustration

	A	Message	B	Comments
1	→	< request 8 buffers>	→	A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	B grants messages 0-3 only
3	→	<seq = 0, data = m0>	→	A has 3 buffers left now
4	→	<seq = 1, data = m1>	→	A has 2 buffers left now
5	→	<seq = 2, data = m2>	• • •	Message lost but A thinks it has 1 left
6	←	<ack = 1, buf = 3>	←	B acknowledges 0 and 1, permits 2-4
7	→	<seq = 3, data = m3>	→	A has 1 buffer left
8	→	<seq = 4, data = m4>	→	A has 0 buffers left, and must stop
9	→	<seq = 2, data = m2>	→	A times out and retransmits
10	←	<ack = 4, buf = 0>	←	Everything acknowledged, but A still blocked
11	←	<ack = 4, buf = 1>	←	A may now send 5
12	←	<ack = 4, buf = 2>	←	B found a new buffer somewhere
13	→	<seq = 5, data = m5>	→	A has 1 buffer left
14	→	<seq = 6, data = m6>	→	A is now blocked again
15	←	<ack = 6, buf = 0>	←	A is still blocked
16	• • •	<ack = 6, buf = 4>	←	Potential deadlock

Figure 6-16. Dynamic buffer allocation. The arrows show the direction of transmission. An ellipsis (...) indicates a lost segment.

Figure 7: Buffer Size illustration

Situation at line 16 can be handled like: To prevent this situation, each host should periodically send control segments giving the acknowledgement and buffer status on each connection. That way, the deadlock will be broken, sooner or later.

When buffer space no longer limits the maximum flow, another bottleneck will appear: the carrying capacity of the network. If adjacent routers can ex- change at most x packets/sec and there are k disjoint paths between a pair of hosts, there is no way that those hosts can exchange more than kx segments/sec, no mat- ter how much buffer space is available at each end. If the sender pushes too hard (i.e., sends more than kx segments/sec), the network will become congested be- cause it will be unable to deliver segments as fast as they are coming in. Belsnes (1975) proposed using a sliding window flow-control scheme in which the sender dynamically adjusts the window size to match the network's carrying capacity. This means that a dynamic sliding window can implement both flow control and congestion control. If the network can handle c segments/sec and the round-trip time (including transmission, propagation, queueing, proc- essing at the receiver, and return of the acknowledgement) is r, the sender's win- dow should be cr. With a window of this size, the sender normally operates with the pipeline full.

Multiplexing

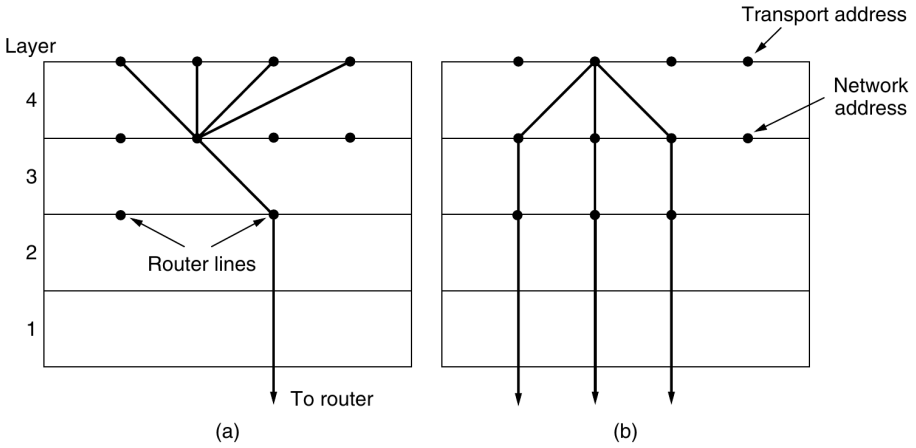


Figure 6-17. (a) Multiplexing. (b) Inverse multiplexing.

Figure 8: MP

In the transport layer, the need for multiplexing can arise in a number of ways. For example, if only one network address is available on a host, all transport con- nections on that machine have to use it. When a segment comes in, some way

is needed to tell which process to give it to. This situation, called multiplexing, is shown in Fig. 6-17(a). In this figure, four distinct transport connections all use the same network connection (e.g., IP address) to the remote host.

Multiplexing can also be useful in the transport layer for another reason. Suppose, for example, that a host has multiple network paths that it can use. If a user needs more bandwidth or more reliability than one of the network paths can pro- vide, a way out is to have a connection that distributes the traffic among multiple network paths on a round-robin basis, as indicated in Fig. 6-17(b). This modus operandi is called inverse multiplexing. With k network connections open, the effective bandwidth might be increased by a factor of k. An example of inverse multiplexing is SCTP (Stream Control Transmission Protocol), which can run a connection using multiple network interfaces.

Crash Recovery

let us assume that one host, the client, is sending a long file to another host, the file server, using a simple stop-and-wait protocol. The transport layer on the server just passes the incoming segments to the transport user, one by one. Partway through the transmission, the server crashes. When it comes back up, its tables are reinitialized, so it no longer knows precisely where it was. In an attempt to recover its previous status, the server might send a broadcast segment to all other hosts, announcing that it has just crashed and requesting that its clients inform it of the status of all open connections. Each client can be in one of two states: one segment outstanding, S1, or no segments outstanding, S0. Based on only this state information, the client must decide whether to retransmit the most recent segment. At first glance, it would seem obvious: the client should retransmit if and only if it has an unacknowledged segment outstanding (i.e., is in state S1) when it learns of the crash. However, a closer inspection reveals difficulties with this naive approach. Consider, for example, the situation in which the server's transport entity first sends an acknowledgement and then, when the acknowledgement has been sent, writes to the application process. Writing a segment onto the output stream and sending an acknowledgement are two distinct events that cannot be done simultaneously. If a crash occurs after the acknowledgement has been sent but before the write has been fully completed, the client will receive the acknowledgement and thus be in state S0 when the crash recovery announcement arrives. The client will therefore not retransmit, (incorrectly) thinking that the segment has arrived. This decision by the client leads to a missing segment.

At this point you may be thinking: "That problem can be solved easily. All you have to do is reprogram the transport entity to first do the write and then send the acknowledgement." Try again. Imagine that the write has been done but the crash occurs before the acknowledgement can be sent. The client will be in state S1 and thus retransmit, leading to an undetected duplicate segment in the output stream to the server application process. No matter how the client and

server are programmed, there are always situations where the protocol fails to recover properly.

Put in more general terms, this result can be restated as "recovery from a layer N crash can only be done by layer N + 1," and then only if the higher layer retains enough status information to reconstruct where it was before the problem occurred. This is consistent with the case mentioned above that the transport layer can recover from failures in the network layer, provided that each end of a connection keeps track of where it is.

Remote Procedure Calls

When a process on machine 1 calls a procedure on machine 2, the calling process on 1 is suspended and execution of the called procedure takes place on 2. Information can be transported from the caller to the callee in the parameters and can come back in the procedure result. No message passing is visible to the application programmer. This technique is known as RPC (Remote Procedure Call) and has become the basis for many networking applications. Traditionally, the calling procedure is known as the client and the called procedure is known as the server

In the simplest form, to call a remote procedure, the client program must be bound with a small library procedure, called the client stub, that represents the server procedure in the client's address space. Similarly, the server is bound with a procedure called the server stub. These procedures hide the fact that the procedure call from the client to the server is not local.

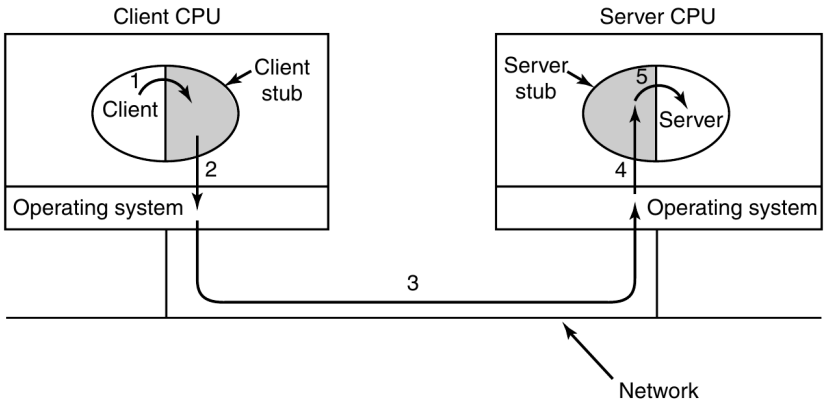


Figure 9: RPC

The actual steps in making an RPC are shown in Fig. 6-29. Step 1 is the client calling the client stub. This call is a local procedure call, with the parameters

pushed onto the stack in the normal way. Step 2 is the client stub packing the parameters into a message and making a system call to send the message. Packing the parameters is called marshaling. Step 3 is the operating system sending the message from the client machine to the server machine. Step 4 is the operating system passing the incoming packet to the server stub. Finally, step 5 is the server stub calling the server procedure with the unmarshaled parameters. The reply traces the same path in the other direction.

Problems

- Pointers cannot be copy because they lie in different address space.
- Arrays cant be marshalled because their size is unknown.
- Arguments type and number can be any.
- Global variables.
- Operation may not be idempotent (i.e., safe to repeat). (In case packet is to be resent)